21世纪高等学校规划教材 | 软件工程





软件测试

韩利凯 主编
高寅生 袁溪 副主编

清华大学出版社

21世纪高等学校规划教材·软件工程

软件测试

韩利凯　主编
高寅生　袁　溪　副主编

清华大学出版社
北京

内 容 简 介

本书针对高校计算机相关专业软件测试课程的需要而编写，系统地介绍了软件测试的基础知识与应用技术。本书内容包括软件测试的基本概念和基本知识、软件测试计划、软件测试的基本技术、软件测试过程、测试用例设计、测试报告与测试评测、软件测试项目管理、面向对象软件测试、软件测试自动化以及一个实际软件项目的测试案例，通过该案例的学习，以加深读者对软件测试技术和软件测试过程的理解，加强理论知识的实践性。本书还对目前比较流行的测试工具软件做了介绍。

本书内容全面、深入浅出、理论和实践相结合，通过本书的学习读者能够较好地掌握软件测试的基本知识和基本技术。本书可作为高校计算机专业的软件测试课程的教材，也可作为软件测试培训班的教材或者软件测试人员的自学参考书。

图书在版编目(CIP)数据

软件测试/韩利凯主编. —北京：清华大学出版社，2013
21 世纪高等学校规划教材·软件工程
ISBN 978-7-302-32840-7

Ⅰ. ①软…　Ⅱ. ①韩…　Ⅲ. ①软件－测试－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2013)第 136596 号

责任编辑：郑寅堃　王冰飞
封面设计：傅瑞学
责任校对：梁　毅
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张**：12.75　**字　　数**：320 千字
版　　次：2013 年 8 月第 1 版　**印　　次**：2013 年 8 月第1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：049685-01

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施"高等学校本科教学质量与教学改革工程(简称'质量工程')"，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程(群)加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合"质量工程"的实施，满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的"高等学校教学质量与教学改革工程精品课程建设工作"，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了"清华大学出版社教材编审委员会"(以下简称"编委会")，旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。"编委会"成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，"编委会"一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上；精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性(新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量)、先进性(对原有的学科体系有实质性的改革和发展，顺应并符合 21 世纪教学发展的规律，代表并引领课程发展的趋势和方向)、示范性(教材所体现的课程体系具有较广泛的辐射性和示范性)和一定的前瞻性。教材由个人申报或各校推荐(通过所在高校的"编委会"成员推荐)，经"编委会"认真评审，最后由清华大学出版

社审定出版。

目前，针对计算机类和电子信息类相关专业成立了两个“编委会”，即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。推出的特色精品教材包括：

（1）21世纪高等学校规划教材·计算机应用——高等学校各类专业，特别是非计算机专业的计算机应用类教材。

（2）21世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

（3）21世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

（4）21世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

（5）21世纪高等学校规划教材·信息管理与信息系统。

（6）21世纪高等学校规划教材·财经管理与应用。

（7）21世纪高等学校规划教材·电子商务。

（8）21世纪高等学校规划教材·物联网。

清华大学出版社经过三十多年的努力，在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌，为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格，这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会
联系人：魏江江
E-mail：weijj@tup.tsinghua.edu.cn



前 言

随着软件规模和复杂性的大幅度提升,软件质量可靠性的问题变得日益突出。软件测试是保证软件质量的关键技术之一,同时也是软件开发过程中的一个重要环节,其理论知识和技术工具都在飞速发展。几乎每个大中型IT企业的软件产品在发布前都需要大量的质量控制、测试和文档工作,而这些工作必须依靠拥有娴熟技术的专业软件人才来完成。软件测试人员就是这样的一个企业重头角色。但是目前在国内大多数软件企业中,测试人员的数量不到开发人员数量的五分之一,远远落后于国外先进水平。由此可见,我国对软件测试人员的需求量越来越大。

在上述社会现状下,高校计算机相关专业已广泛开设软件测试课程,由此,对软件测试课程教材的需求也与日俱增,特别是理论和实践相结合的优质教材。

本书的编写特色是:(1)理论和实践相结合,在理论基础上拓展实践技能。本书前半部分详细介绍软件测试理论知识,后半部分通过一个实际软件项目的测试案例将理论知识加以应用,以提高读者的实践能力。(2)注重知识的多元性。全书贯穿软件工程、软件质量管理和软件项目管理等知识,让读者体会到软件测试在整个软件工程生命周期中的地位和作用。(3)例题丰富。在软件测试基本技术、软件测试用例设计等章节有详细的例题分析,有助于读者对理论知识的深刻理解。

本书按照软件测试流程编写。第1章主要介绍软件测试的基本概念和相关理论;第2章详细讲解软件测试计划,强调在开始测试工作之前,详细、全面的计划是很重要的;第3~5章详细介绍软件测试的核心技术和测试过程,以及测试用例的设计方法,还重点介绍白盒测试法、黑盒测试法、单元测试、集成测试、系统测试、回归测试等重点知识和相关技能,同时针对测试用例的设计给出若干例题,让读者全面理解软件测试的实际测试方法;第6章、第7章具体介绍软件测试项目跟踪和管理的相关知识和技术,以培养读者软件测试文档的编写、缺陷的报告和分析以及问题跟踪系统等各方面的技能;第8章介绍面向对象软件的测试方法,因为面向对象软件具有自身的特殊性,所以单独对它进行讲解;第9章重点介绍软件测试自动化的引入、发展和优缺点,让读者能正确地认识软件测试自动化,并介绍目前较为流行的一些测试工具,包括LoadRunner、WinRunner、TestDirector、JUnit等;第10章通过一个实际软件项目的测试案例,把软件测试的流程系统地展现在读者面前,加深读者对软件测试技术和软件测试过程的理解,使其学会理论和实践的结合。

本书由韩利凯主编,高寅生、袁溪任副主编。参与编写的人员有韩利凯、袁溪、孙少波、袁佳乐、任强、吴燕妮。编写安排如下:孙少波编写第1章、第7章;袁佳乐编写第2章、第5章;韩利凯编写第3章;任强编写第4章;袁溪编写第6章、第9章、第10章;吴燕妮编写第8章。本书在编写过程中得到了多方面的帮助、指导和支持,在此向他们表示由衷的感谢。

由于编者水平有限,书中难免有疏漏和不妥之处,敬请广大读者提出宝贵的意见和建议。

编　者

2013年5月于西安文理学院

目 录

第1章 软件测试概述

软件的质量就是软件的生命，人们在长期的开发过程中积累了大量经验并形成了许多行之有效的方法，以期保证软件的质量。但是借助这些方法，我们也只能尽量减少软件中的错误和不足，却不能完全避免错误。软件测试是最有效的排除和防止软件缺陷与故障的手段，并由此促进了软件测试理论与技术的快速发展。新的测试理论、测试方法、测试技术手段正在不断涌现。

本章主要介绍软件测试的一些基础知识：软件危机和软件工程、软件缺陷与软件故障、软件质量与质量模型、软件测试、软件测试人员的基本素质等相关知识。

1.1 软件、软件危机和软件工程

1.1.1 软件

软件是一系列按照特定顺序组织的计算机数据和指令的集合。一般来讲软件被划分为编程语言、系统软件、应用软件和介于后两者之间的中间件。软件并不只包括可以在计算机(这里的计算机是指广义的计算机)上运行的计算机程序，与这些计算机程序相关的文档一般也被认为是软件的一部分。简单地说软件就是程序加文档的集合体。

1. 系统软件

系统软件是为计算机使用提供最基本的功能的软件，它负责管理计算机系统中各种独立的硬件，使得它们可以协调工作。系统软件使得计算机使用者和其他软件可以将计算机当作一个整体而不需要顾及底层每个硬件是如何工作的。系统软件可分为操作系统和支撑软件，其中操作系统是最基本的软件。

(1) 操作系统是管理计算机硬件与软件资源的程序，它是计算机系统的内核与基石。操作系统身负诸如管理与配置内存、决定系统资源供需的优先次序、控制输入与输出设备、操作网络与管理文件系统等基本事务，并提供一个让使用者与系统交互的操作接口。

(2) 支撑软件顾名思义是支撑各种软件开发与维护的软件，又称为软件开发环境(SDE)。它主要包括环境数据库、各种接口软件和工具组。著名的软件开发环境有 IBM 公司的 Web Sphere 等，包括一系列基本的工具(例如编译器、数据库管理、存储器格式化、文件系统管理、用户身份验证、驱动管理、网络连接等方面的工具)。

2. 应用软件

系统软件并不针对某一特定应用领域,而应用软件则相反,不同的应用软件根据用户和所服务的领域提供不同的功能。

简而言之,应用软件是为了某种特定的用途而被开发的软件。它可以是一个特定的程序,比如一个图像浏览器;也可以是一组功能联系紧密,可以互相协作的程序的集合,比如微软的Office软件;还可以是一个由众多独立程序组成的庞大的软件系统,比如数据库管理系统。

3. 手机软件

顾名思义,手机软件就是可以安装在手机上的软件,用于完善原始系统的不足并提供个性化。随着科技的发展,现在手机的功能也越来越多,越来越强大,不像过去那么简单死板,目前手机甚至发展到了可以和掌上电脑相媲美。手机软件与计算机软件一样,要考虑你购买这一款手机所安装的系统来决定下载相对应的软件。目前主流手机系统有以下几种:Windows Phone、Symbian、Linux、Research in Motion、Windows Mobile、iPhone iOS、Android。

1.1.2 软件危机

20世纪60年代以前,由于计算机刚刚投入实际使用,软件设计往往只是为了一个特定的应用而在指定的计算机上设计和编制,采用密切依赖于计算机的机器代码或汇编语言,软件的规模比较小,文档资料通常也不存在,很少使用系统化的开发方法,设计软件往往等同于编制程序,基本上是个人设计、个人使用、个人操作、自给自足的私人化的软件生产方式。

20世纪60年代中期,大容量、高速度计算机的出现,使计算机的应用范围迅速扩大,软件开发急剧增长。高级语言开始出现,操作系统的发展引起了计算机应用方式的变化,大量数据处理导致第一代数据库管理系统的诞生。软件系统的规模越来越大,复杂程度越来越高,软件可靠性问题也越来越突出。原来的个人设计、个人使用的方式不再能满足要求,迫切需要改变软件生产方式,提高软件生产率。

1968年北大西洋公约组织的计算机科学家在联邦德国召开国际会议,第一次讨论软件的危机问题。所谓软件危机(software crisis),泛指在计算机软件的开发和维护过程中所遇到的一系列严重问题。

1. 软件危机的主要表现

软件危机的主要表现如下。

(1) 软件开发费用和进度失控。费用超支、进度拖延的情况屡屡发生。有时为了赶进度或压低成本不得不采取一些权宜之计,这样又往往严重损害了软件产品的质量。

(2) 软件的可靠性差。尽管耗费了大量的人力物力,而系统的正确性却越来越难以保证,出错率大大增加,由于软件错误而造成的损失十分惊人。

(3) 生产出来的软件难以维护。很多程序缺乏相应的文档资料,程序中的错误难以定位,难以改正,有时改正了已有的错误又引入新的错误。随着软件的社会拥有量越来越大,维护占用了大量人力、物力和财力。进入20世纪80年代以来,尽管软件工程研究与实践取得了可喜的成就,软件技术水平有了长足的进展,但是软件生产水平依然远远落后于硬件生

产水平的发展速度。

(4) 用户对“已完成”的系统不满意现象经常发生。一方面,许多用户在软件开发的初期不能准确完整地向开发人员表达他们的需求;另一方面,软件开发人员常常在对用户需求还没有正确全面认识的情况下,就急于编写程序。

(5) 软件成本在计算机系统总成本中所占的比例居高不下,且逐年上升。由于微电子学技术的进步和硬件生产自动化程度不断提高,硬件成本逐年下降,性能和产量迅速提高。然而软件开发需要大量人力,软件成本随着软件规模和数量的剧增而持续上升。美、日两国的统计数字表明,1985 年度软件成本甚至占到计算机系统总成本的将近 90%。

(6) 软件开发生产率提高的速度远远跟不上计算机应用迅速普及深入的需要,软件产品供不应求的状况使得人类不能充分利用现代计算机硬件所能提供的巨大潜力。

2. 产生软件危机的原因

研究结果表明,产生软件危机的原因主要有以下几个方面。

1) 与软件本身的特点有关

软件不同于硬件,它是计算机系统中的逻辑部件:软件样品即是产品,试制过程也就是生产过程;软件不会因使用时间过长而“老化”或“用坏”;软件具有可运行的行为特性,在写出程序代码并在计算机上试运行之前,软件开发过程的进展情况较难衡量,软件质量也较难评价,因此管理和控制软件开发过程十分困难;软件质量不是根据大量制造的相同实体的质量来度量的,而是与每一个组成部分的不同实体的质量紧密相关,因此,在运行时所出现的软件错误几乎都是在开发时期就存在而一直未被发现的,改正这类错误通常意味着改正或修改原来的设计,这就在客观上使得软件维护远比硬件维护困难;软件是一种信息产品,具有可延展性,属于柔性生产,与通用性强的硬件相比,软件更具有多样化的特点,更加接近人们的应用问题。

2) 用户需求不明确

在软件开发过程中,用户需求不明确问题主要体现在 4 个方面:

- 在软件开发出来之前,用户自己也不清楚软件开发的具体需求;
- 用户对软件开发需求的描述不精确,可能有遗漏、有二义性,甚至有错误;
- 在软件开发过程中,用户还提出修改软件开发功能、界面、支撑环境等方面的要求;
- 软件开发人员对用户需求的理解与用户本来愿望有差异。

3) 缺乏正确的理论指导

缺乏有力的方法学和工具方面的支持。由于软件开发不同于大多数其他工业产品,其开发过程是复杂的逻辑思维过程,其产品在很大程度上依赖于开发人员的智力投入。由于过分地依靠程序设计人员在软件开发过程中的技巧和创造性,加剧软件开发产品的个性化,也是发生软件危机的一个重要原因。

4) 软件开发规模越来越大

随着软件开发应用范围的增广,软件开发规模愈来愈大。大型软件开发项目需要组织一定的人力共同完成,而多数管理人员缺乏开发大型软件开发系统的经验,多数软件开发人员又缺乏管理方面的经验。各类人员的信息交流不及时、不准确、有时还会产生误解。软件开发项目开发人员不能有效地、独立自主地处理大型软件开发的全部关系和各个分支,因此

容易产生疏漏和错误。

5）软件开发复杂度越来越高

软件不仅仅在规模上快速地发展扩大，而且其复杂性也急剧地增加。软件开发产品的特殊性和人类智力的局限性，导致人们无力处理“复杂问题”。所谓“复杂问题”的概念是相对的，一旦人们采用先进的组织形式、开发方法和工具提高了软件开发效率和能力，新的、更大的、更复杂的问题又摆在人们的面前。

1.1.3 软件工程

在 1968 年的北约计算机科学家国际会议上，同时提出了“软件工程”一词，正式宣告软件工程学的诞生，作为一个新兴的工程学科，软件工程主要研究软件生产的客观规律性，建立与系统化软件生产有关的概念、原则、方法、技术和工具，指导和支持软件系统的生产活动，以期达到降低软件生产成本、改进软件产品质量、提高软件生产率水平的目标。软件工程学从硬件工程和其他人类工程中吸收了许多成功的经验，明确提出了软件生命周期的模型，发展了许多软件开发与维护技术和方法，并应用于软件工程实践，取得良好的效果。

在不断演进的软件开发过程中人们开始研制和使用软件工具，用以辅助进行软件项目管理与技术生产，人们还将软件生命周期各阶段使用的软件工具有机地集合成为一个整体，形成能够连续支持软件开发与维护全过程的集成化软件支援环境，以期从管理和技术两方面解决软件危机问题。

软件工程学科诞生后，人们为其给出了许多不同的定义。最早的定义是由 F. L. Bauer 给出的，即“软件工程是为了经济地获得能够在实际机器上高效运行的、可靠的软件而建立和应用一系列坚实的软件工程原则”。美国梅隆卡耐基大学软件工程研究所（SEI）给出的定义则是：软件工程是以工程的形式应用计算机科学和数学原理，从而经济有效地解决软件问题。但目前普遍使用的软件工程定义是由 IEEE 给出的，即软件工程是将系统性的、规范化的、可定量的方法应用于软件的开发、运行和维护。

综合来看，软件工程概念实际存在两层含义，从狭义概念看，软件工程着重体现在软件过程中所采用的工程方法和管理体系，例如，引入成本核算、质量管理和项目管理等，即将软件产品开发看作是一项工程项目所需要的系统工程学和管理学。从广义概念看，软件工程涵盖了软件生命周期中所有的工程方法、技术和工具，包括需求工程、设计、编程、测试和维护的全部内容，即完成一个软件产品所必备的思想、理论、方法、技术和工具。

软件工程学科包含为完成软件需求、设计、构建、测试和维护所需的知识、方法和工具。对软件工程的理解不应局限在理论之上，要更注重实践，软件工程能够帮助软件组织协调团队、运用有限的资源，遵守已定义的软件工程规范，通过一系列可复用的、有效的方法，在规定的时间内达到预先设定的目标。针对软件工程的实施，无论是采用什么样的方法和工具，先进的软件工程思想始终是最重要的。只有在正确的工程思想指导下，才能制定正确的技术路线，才能正确地运用方法和工具达到软件工程或项目管理的既定目标。

软件工程是一门交叉性的工程学科，它将计算机科学、数学、工程学和管理学等基本原理应用于软件的开发与维护中，其重点在于大型软件的分析与评价、规格说明、设计和演化，同时涉及管理、质量、创新、标准、个人技能、团队协作和专业实践等。从这个意义上看，软件工程可以看作由下列 3 部分组成。

- 计算机科学和数学，用于构造软件的模型与算法。
- 工程科学，用于制定规范、设计范型、评估成本以及确定权衡等。
- 管理科学，用于计划、资源、质量、成本等管理。

例如，计算机辅助软件工程（Computer Aided Software Engineering，CASE）就是一组工具和方法的集合，可以辅助软件生命周期各阶段进行的软件开发活动。CASE 吸收了 CAD（计算机辅助设计）、软件工程、操作系统、数据库、网络和许多其他计算机领域的原理和技术。这个例子也体现了软件工程是学科交叉的、集成和综合的领域。

如果从知识领域看，软件工程学科是以软件方法和技术为核心，涉及计算机的硬件体系、系统基础平台等相关领域，同时还涉及一些应用领域和通用的管理学科、组织行为学科。例如，通过应用领域的知识帮助我们理解用户的需求，从而可以根据需求来设计软件的功能。在软件工程中必然要涉及组织中应用系统的部署和配置所面临的实际问题，同时又必须不断促进知识的更新和理论的创新。为了真正解决实际问题，需要在理论和应用上获得最佳平衡。

1.2　软件缺陷与软件故障

1.2.1　相关概念

质量、进度和成本是软件项目关注的 3 大要素，它们相互依赖、相互制约，软件项目管理就是要在这 3 个目标的建立、跟踪和实现方面达成最佳均衡。软件质量要求的高低，既会影响到成本，又会影响到进度。一般来说，不论是顾客还是承制方，对软件质量的既定目标都不会妥协，宁可追加成本或延迟进度，也不会迁就质量目标的降低，对质量都采用“一票否决制”，因此，软件质量在三大目标中往往是位于第一位的。不论是业界传统的狭义软件质量观，还是以 ISO/IEC 9126 系列标准代表的广义软件质量观，软件缺陷的概念都在软件质量范畴中处于举足轻重的地位。在狭义软件质量观中，软件缺陷的概念绝对处于核心地位，它衍生出一系列软件质量指标，是考察软件质量的唯一依据。在当今业界大力实施的软件能力成熟度模型（Capability Maturity Model for Software，CMM）或能力成熟度模型集成（Capability Maturity Model Integration，CMMI）中，软件缺陷度量仍然是一个核心度量，尤其是攀登高成熟度级别不可或缺的基石。总之，软件缺陷度量的体系结构是各种软件过程改进模型的基础设施，是实施与评估软件质量活动的先决条件。建立完善的软件缺陷度量的体系结构对软件质量管理的策划和实施、软件缺陷的跟踪和管理、软件质量目标的设定和实现、软件过程能力基线的创建和优化都具有重大的现实意义。

历史上，曾经对与软件缺陷相关的一些概念搞得很混淆，各家众说纷纭，使得从业人员无所适从。下面对这些概念进行区分，它们是建立软件缺陷度量的体系结构的先决条件。

(1) 错误(error)：即人为错误，指软件开发人员在开发软件的过程中无意间犯下的技术错误，正是这些错误导致软件工作产品的缺陷。

(2) 缺陷(defect)：指软件工作产品中不满足指定要求的成分，它是静态的，如果不将其消除，它将永远存在。在业界，人们常用另外一个词“bug”指代缺陷，早期美国海军在调试软件时曾发生一个臭虫(bug)引发系统不能正常工作的情况，此典故流传下来致使人们

将“bug”作为缺陷的代名词。将缺陷俗称为“bug”，易使人们对缺陷轻描淡写，忽视了缺陷的严重性，这是值得注意的问题。缺陷是造成软件故障乃至失效的内在原因。

(3) 故障(fault)：指软件运行时丧失了在规定的限度内执行所需功能的能力，它是动态的，它可能导致失效。故障不一定导致软件失效，软件运行可以出现故障但不出现失效，如在容错(fault tolerance)软件运行中容许有规定数量的故障出现而不导致失效。对无容错的软件，故障即失效。故障是软件缺陷的外在表现。

(4) 失效(failure)：指软件运行时不能完成规定功能，它是动态的，由故障所导致。失效是软件缺陷的外在表现。

图 1-1 演示了以上描述的软件失效机制。

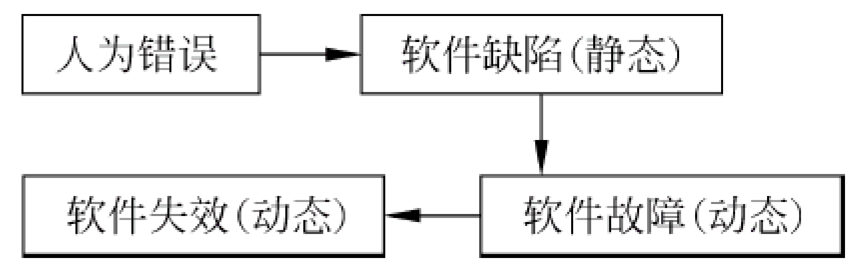


图 1-1　软件失效机制示意图

1.2.2　软件缺陷

1. 软件缺陷的分类分级

历史上，出于对软件缺陷管理的需要，人们曾经对软件缺陷进行过分类分级，如 IBM 等软件开发大公司都制定了内部的软件缺陷分类分级的企业标准，企业标准主要依据本企业的实际情况制定，目的是为本企业所用。另外，还出现过一些推荐性的工业标准，如国军标 GJB 2255－1994。但是从未出现过权威且实用、系统的软件缺陷分类分级的标准，就是大多数企业标准或推荐性的工业标准也都主要关注软件代码的缺陷分类分级，没有覆盖其他主要软件工作产品的缺陷情况。

正交缺陷分类(Orthogonal Defect Classification，ODC)法是一种缺陷分析方法，该方法覆盖软件需求、软件设计和软件代码等主要软件工作产品，使 ODC 法划分的软件缺陷类型如表 1-1 所示。

软件缺陷类别用于区分软件缺陷的性质，对软件缺陷类别的划分如表 1-2 所示。

不同软件缺陷造成软件故障和软件失效的严重程度也不一样，对用户带来的危害和损失程度不一样，因此有必要区分软件缺陷的严重等级，对软件缺陷严重等级的划分见表 1-3。

表 1-1　软件缺陷类型

大类号	大类名称	小类号	小类名称
1	需求缺陷 (软件需求不满足顾客的要求)	1	功能
		2	性能
		3	接口
		4	控制流
		5	数据流
		6	标准
		7	一致性
		8	可追溯性
		9	文档版本
		10	其他

续表

大类号	大类名称	小类号	小类名称
2	设计缺陷（软件设计不满足软件需求规格说明的要求）	1	功能
		2	性能
		3	接口
		4	逻辑
		5	数据使用
		6	错误处理
		7	标准
		8	一致性
		9	可追溯性
		10	文档版本
		11	其他
3	代码缺陷（软件代码不满足软件设计说明的要求）	1	功能
		2	性能
		3	接口
		4	逻辑
		5	数据使用
		6	错误处理
		7	编程语言
		8	编程规范
		9	代码冗余
		10	可追溯性
		11	代码版本
		12	其他

表 1-2　软件缺陷类别

序号	类别名称	说明
1	遗漏	应有的内容缺失
2	错误	相应的内容错误
3	多余	不应有的内容出现

表 1-3　软件缺陷严重等级

序号	等级名称	说明
1	致命	导致软件崩溃、死机的缺陷
2	严重	妨碍主要功能实现和性能达标的缺陷
3	一般	妨碍次要功能实现的缺陷
4	轻微	给用户带来不方便或麻烦，但是不妨碍功能实现的缺陷

2. 软件缺陷的生存周期

软件缺陷的生存周期是指软件缺陷从产生到消除的持续时间。当由于人工失误将缺陷

注入(inject)软件工作产品中,在软件产品发布前发现软件缺陷的主要方法是软件评审和软件测试。在软件生存周期中,各个阶段注入的软件缺陷被发现和消除(remove)的时机包括本阶段以及此后的所有阶段,每个阶段注入的软件缺陷在本阶段被发现和消除是最佳的选择,因为前期阶段注入的软件缺陷在后面阶段被发现和消除的代价比在本阶段被发现和消除所花的代价高得多,势必造成对成本目标实现不利的风险。

在软件生存周期的每个阶段,都要进行软件质量控制,目标是不但要尽量发现和消除本阶段的软件缺陷,还要尽量发现和消除以前所有阶段遗留下来的软件缺陷。软件生存周期中软件缺陷的注入、发现和消除进程如图 1-2 所示。

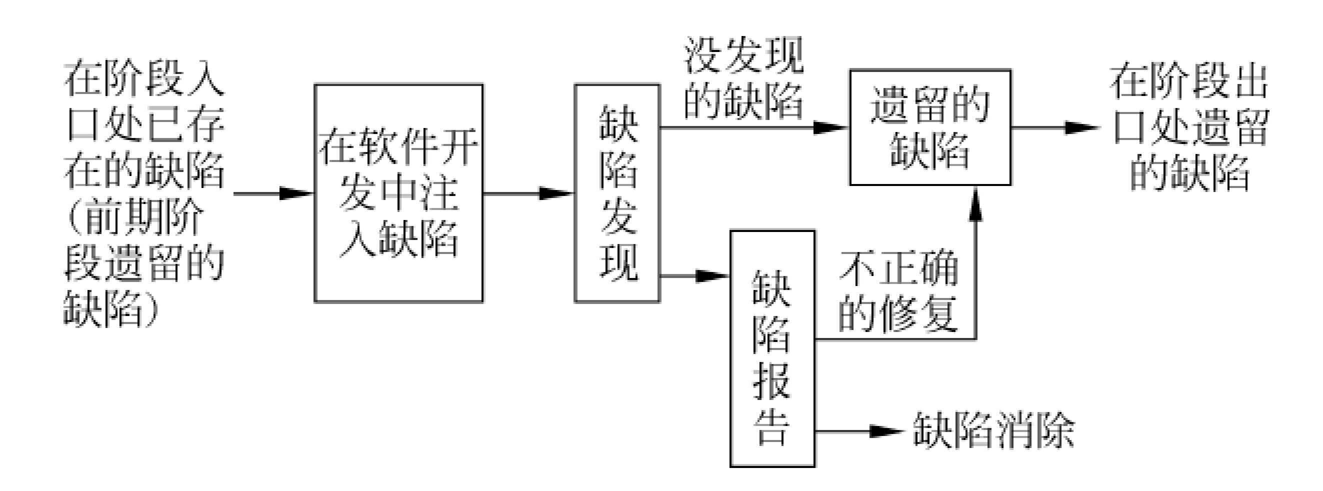


图 1-2 软件缺陷注入、发现和消除过程

1.2.3 软件故障

软件一旦出现故障,有可能造成巨大的危害。例如,1996 年 Ariane 5 运载火箭的发射失败便是软件故障问题带来的一个深刻教训。

1. 软件故障的定义和分类

软件故障定义主要有以下几种:

- 软件故障定义为计算机程序中不正确的步骤、处理或者数据定义。
- 软件故障定义为软件运行过程中出现的一种不希望或不可接受的内部状态。
- 软件故障分为语法大小和语义大小,语法大小为受一个故障影响的代码行数,语义大小为其输出不正确的输入空间的大小。
- 软件故障定义为软件系统中的结构不完善,它可能导致系统的最终失效。
- 软件故障模型是软件物理错误的抽象,是一些基本故障的组合。

从不同的角度,软件故障可以分为如下的类别:

- 根据故障发生阶段分类,有需求分析错误、概要设计错误、详细设计错误、编码错误等。
- 根据故障引起后果分类,有小错误、中等错误、较严重错误、严重错误、非常严重错误、最严重错误。
- 根据故障性质分类,有需求错误、功能和性能错误、结构错误、数据错误、实现和编码错误、集成错误、测试错误等。
- 根据故障类型分类,有文档错误、语法错误、联编打包错误、赋值错误、接口错误、数据错误、函数错误、系统错误、环境错误等。

2. 软件故障的诊断

软件故障诊断是根据软件(包括程序、数据和文档)的静态表现形式和动态运行状态信息查找故障源,并确定相应决策的一门技术。

在参与软件生存周期的各个阶段工作时难免会出现错误。因此,从广义上说,软件故障诊断的目标包括软件需求分析、设计、编码、测试、使用、维护等各阶段所造成的缺陷,软件评审属于软件故障诊断的手段。

软件故障诊断的"诊"在于进行客观的状态检测,包括采用各种测量、分析和诊别方法;"断"则需要确定软件故障特性、软件故障模式、软件故障部位,以及说明软件故障产生的原因,并且提出相应的纠正措施和预防措施等,这是软件诊断技术的关键。软件故障诊断突出了诊断的目的性,即寻找和发现软件故障状态而进行诊断,也包括无故障状态在内,但要强调故障状态的重要性。

软件故障诊断的过程包括故障检测、故障定位、故障排除、回归测试、系统测试和交付等几个阶段。

软件故障检测是软件故障诊断的第一步,可通过静态检查、动态运行等方法获取软件中的各种信息,获得可能出现软件故障的征兆,识别软件是否正常运行或存在故障,并为软件故障定位提供依据。

软件故障定位,是指根据软件故障检测提供的能反映软件状况的征兆或特征参数的变化情况,或与某故障状态参数(模式)进行比较,并进一步收集软件的历史和使用信息,识别软件是否正常运行或存在故障,复现软件故障过程,诊断软件故障的性质和程度、产生原因或发生部位,确定缺陷,为纠正缺陷、排除软件故障做好准备。

软件故障排除是指当诊断出软件中存在缺陷,究其原因、部位和危险程度进行研究,决定纠正缺陷、排除故障的办法,包括修改程序代码、数据或软件文档等。软件故障排除属于软件维护的范畴。

一般来说,在工程应用中进行软件故障诊断的前提是:系统故障经分析、检测确认不是由硬件故障引起,或重点怀疑是由软件故障引起。盲目地进行软件故障诊断将影响系统故障诊断效率。系统出现复杂故障时,也可结合硬件检测,同时进行软件静态检测,这样可以提高系统故障诊断速度。

1.3 软件质量与质量模型

1.3.1 软件质量的定义

关于软件质量,也有许多定义。

1979年,Fisher 和 Light 将软件质量定义为:表征计算机系统卓越程度的所有属性的集合。

1982年,Fisher 又和 Baker 将软件质量定义为:软件产品满足明确需求一组属性的集合。

20世纪90年代,Norman 和 Robin 等将软件质量定义为:一组特性或特征的集合,用

以表征软件产品满足明确的和隐含的需求的能力。

1994 年,国际标准化组织公布的国际标准 ISO 8042 综合将软件质量定义为:反映实体满足明确的和隐含的需求的能力的特性的总和。

综上所述,软件质量是产品、组织和体系或过程的一组固有特性,反映它们满足顾客和其他相关方面要求的程度。如卡耐基梅隆大学软件工程学院(CMU SEI)的 Watts Humphrey 指出:“软件产品必须提供用户所需的功能,如果做不到这一点,什么产品都没有意义。其次,这个产品能够正常工作。如果产品中有很多缺陷,不能正常工作,那么不管这种产品性能如何,用户也不会使用它。”而 Peter Denning 强调:“越是关注客户的满意度,软件就越有可能达到质量要求。程序的正确性固然重要,但不足以体现软件的价值。”

GB/T 11457—2006《软件工程术语》中定义软件质量为:

(1) 软件产品中能满足给定需要的性质和特性的总体;

(2) 软件具有所期望的各种属性的组合程度;

(3) 顾客和用户觉得软件满足其综合期望的程度;

(4) 确定软件在使用中将满足顾客预期要求的程度。

1.3.2 软件质量的三种模型

1. Bohm 质量模型

Bohm 质量模型是由 Bohm 等于 1976 年提出的分层方案,将软件的质量特性定义成分层模型,如图 1-3 所示。

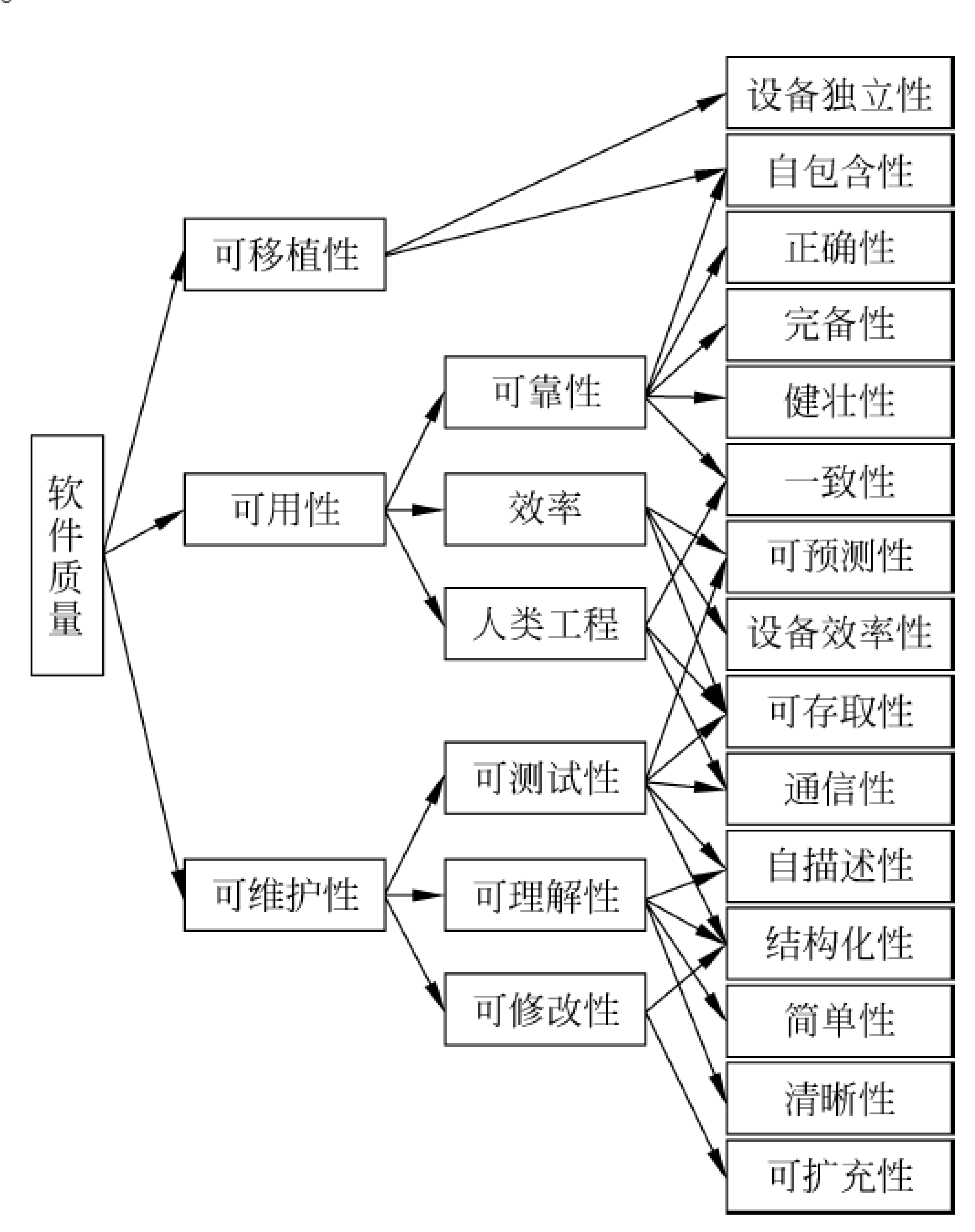


图 1-3 Bohm 质量模型

2. McCall 质量模型

McCall 质量模型是 1979 年由 McCall 等人提出的软件质量模型。它将软件质量的概念建立在 11 个质量特性之上，这些质量特性分别面向软件产品的运行、修正和转移，具体如图 1-4 所示。

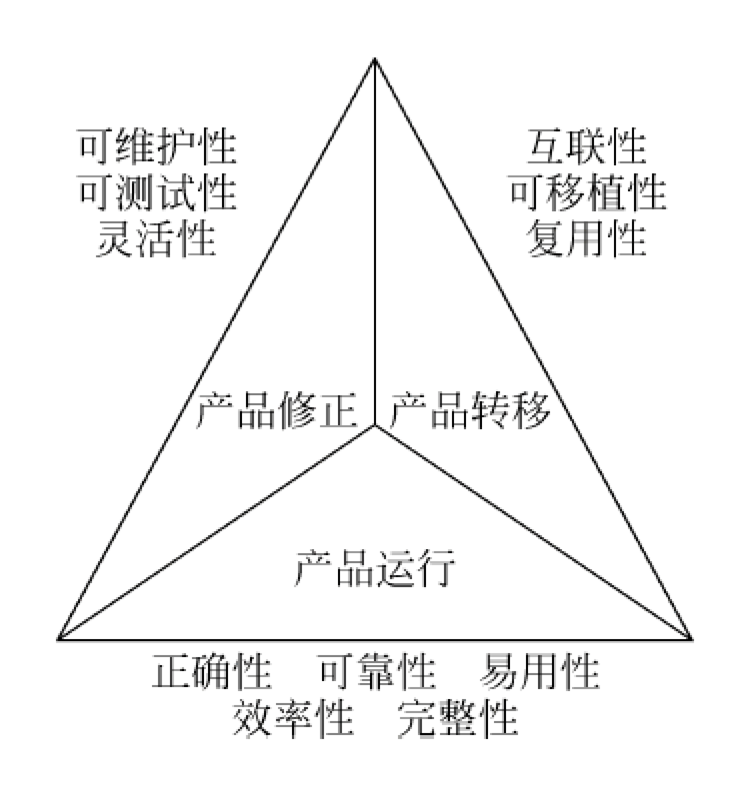


图 1-4 McCall 质量模型

3. ISO 的软件质量模型

按照 ISO/IEC 9126—1：2001，软件质量模型可以分为：内部质量和外部质量模型、使用质量模型，而内部和外部质量又分成六个质量特性，使用质量则包含四个质量属性，具体如图 1-5 和图 1-6 所示。

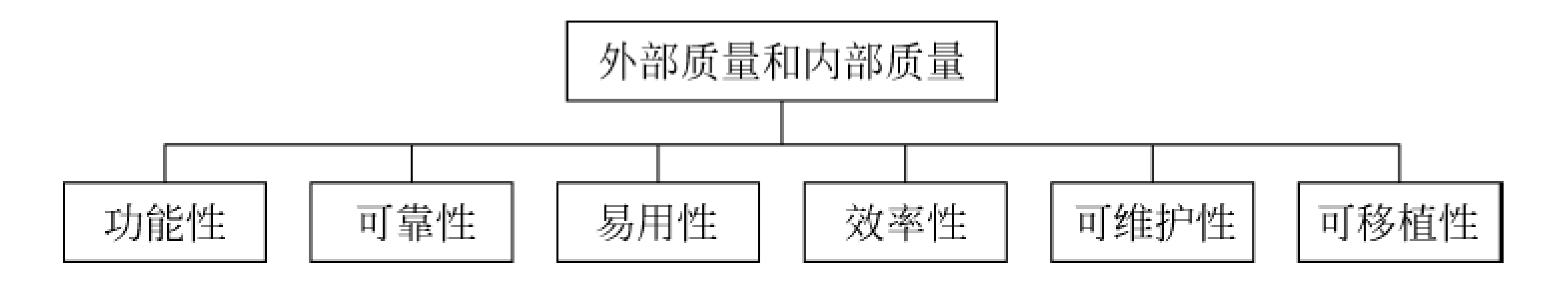


图 1-5 内部质量和外部质量模型

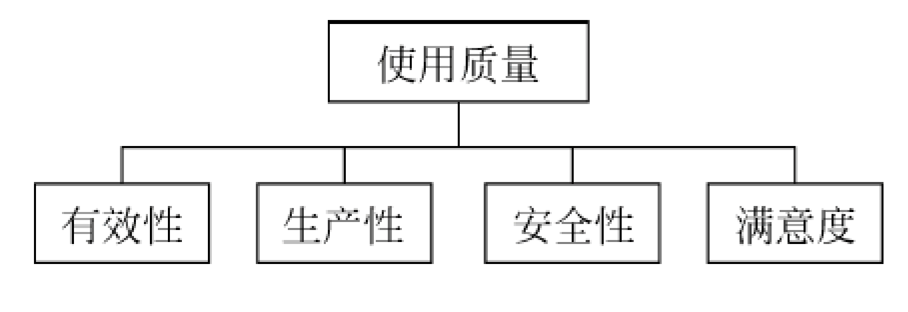


图 1-6 使用质量模型

1.3.3 软件质量的度量

软件质量的度量主要是根据软件生存周期中对软件质量的要求所进行的一项活动。根据 ISO 标准它主要分为三方面：外部度量、内部度量和使用度量。

1. 外部度量

在测试和使用软件产品过程中进行，通过观察该软件产品的系统行为，执行对系统行为

的测量得到度量的结果。

2. 内部度量

在软件设计和编码过程中进行，通过对中间产品的静态分析来测量其内部质量特性。内部度量主要目的是确保获得所需的外部质量和使用质量，与外部关系二者相辅相成，密不可分。

3. 使用质量度量

这是在用户使用过程中完成的，因为使用质量是从用户观点出发对软件产品提出的质量要求，所以它的度量主要是针对用户使用的绩效，而不是软件自身。

1.4 软件测试

所谓软件测试就是利用测试工具按照测试方案和流程对产品进行功能和性能测试，甚至根据需要编写不同的测试工具，设计和维护测试系统，对测试方案可能出现的问题进行分析和评估。执行测试用例后，需要跟踪故障，以确保开发的产品适合需求。

软件测试工程师是负责软件测试的专门工作人员，要求理解产品的功能要求，并对其进行测试，检查软件有没有错误，决定软件是否具有稳定性，写出相应的测试规范和测试用例。简而言之，软件测试工程师在一家软件企业中担当的是"质量管理"角色，负责及时纠错及时更正，确保产品的正常运作。

按其级别和职位的不同，软件测试工程师通常可分为初级软件测试工程师、中级软件测试工程师、高级软件测试工程师三类。要成为一名中级软件测试工程师，你至少需要能够编写软件测试方案、测试文档，与项目组一起制定软件测试阶段的工作计划，能够在项目运行中合理利用测试工具完成测试任务；而高级软件测试工程师则要熟练掌握软件测试与开发技术，且对所测试软件对口行业非常了解，能够对可能出现的问题进行分析评估。软件测试人员必须具有创新性和综合分析能力，必须具备判断准确、追求完美、执著认真、善于合作的品质，以及具有丰富的编程经验与查检故障的能力。

软件测试是使用人工或者自动手段来运行或测试某个系统的过程，其目的在于检验系统是否满足规定的需求或弄清预期结果与实际结果之间的差别。

它是帮助识别开发完成（中间或最终的版本）的计算机软件（整体或部分）的正确度（correctness）、完全度（completeness）和质量（quality）的软件过程；是 SQA（Software Quality Assurance）的重要子域。

Grenford J. Myers 曾对软件测试的目的提出过以下观点：

- 测试是为了发现程序中的错误而执行程序的过程。
- 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案。
- 成功的测试是发现了至今为止尚未发现的错误的测试。
- 测试并不仅仅是为了找出错误。通过分析错误产生的原因和错误的发生趋势，可以帮助项目管理者发现当前软件开发过程中的缺陷，以便及时改进。
- 这种分析也能帮助测试人员设计出有针对性的测试方法，改善测试的效率和有效性。
- 没有发现错误的测试也是有价值的，完整的测试是评定软件质量的一种方法。

另外，根据测试目的的不同，还有回归测试、压力测试、性能测试等，分别用于检验修改或优化过程是否引发新的问题、软件所能达到处理能力和是否达到预期的处理能力等。

1.4.1 软件测试原则

软件测试原则如下：

(1) 测试应该尽早进行，最好在需求阶段就开始介入，因为最严重的错误不外乎是系统不能满足用户的需求。

(2) 程序员应该避免检查自己的程序，软件测试应该由第三方来负责。

(3) 设计测试用例时应考虑到合法的输入和不合法的输入以及各种边界条件，特殊情况下还要制造极端状态和意外状态。

(4) 应该充分注意测试中的群集现象。

(5) 对策是就错误结果进行的一个确认过程。一般由 A 测试出来的错误，一定要由 B 来确认。严重的错误可以召开评审会议进行讨论和分析，对测试结果要进行严格的确认，检查是否真的存在这个问题以及严重程度等。

(6) 制定严格的测试计划。一定要制定测试计划，并且要有指导性。测试时间安排尽量宽松，不要希望在极短的时间内完成一个高水平的测试。

(7) 妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便。

1.4.2 软件测试目标

软件测试目标如下：

(1) 发现一些可以通过测试避免的开发风险。

(2) 实施测试来降低所发现的风险。

(3) 确定测试何时可以结束。

(4) 在开发项目的过程中将测试看作是一个标准项目。

1.4.3 软件测试的具体内容

软件测试主要工作内容是验证和确认，下面分别给出其概念。

1. 验证

验证(verification)是保证软件正确地实现了一些特定功能的一系列活动，即保证软件以正确的方式来做这个事件。它是：

(1) 确定软件生存周期中的一个给定阶段的产品是否达到前阶段确立的需求的过程。

(2) 程序正确性的形式证明，即采用形式理论证明程序符合设计规定的过程。

(3) 评估、审查、测试、检查、审计等各类活动，或对某些项处理、服务或文件等是否和规定的需求相一致进行判断和提出报告。

2. 确认

确认(validation)的目的是想证实在一个给定的外部环境中软件的逻辑正确性。即保

证软件做了你所期望的事情。可以分为以下两种：

（1）静态确认：不在计算机上实际执行程序，通过人工或程序分析来证明软件的正确性。

（2）动态确认：通过执行程序做分析，测试程序的动态行为，以证实软件是否存在问题。

软件测试不仅仅是程序测试，其对象应该包括整个软件开发期间各个阶段所产生的文档，如需求规格说明、概要设计文档、详细设计文档，当然软件测试的主要对象还是源程序。

1.4.4 软件测试的主要方法

下面逐一介绍目前常用的各种软件测试方法，可通过这些方法的介绍了解软件测试的测试内容。

1. 黑盒测试

黑盒测试也称功能测试或数据驱动测试，它是在已知产品所应具有的功能的情况下，通过测试来检测每个功能是否都能正常使用。在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。黑盒测试方法主要有等价类划分、边界值分析、因果图、错误推测等，主要用于软件确认测试。黑盒测试法着眼于程序外部结构，而不考虑内部逻辑结构，针对的是软件接口和软件功能。黑盒测试法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

2. 白盒测试

白盒测试也称结构测试或逻辑驱动测试，它是在知道产品内部工作过程的前提下，检测产品内部动作是否按照规格说明书的规定正常进行。即按照程序内部的结构测试程序，检验程序中的每条通路是否都能按预定要求正确工作，而不顾它的功能。白盒测试的主要方法有逻辑驱动、基路测试等，主要用于软件验证。

白盒测试法要求全面了解程序内部逻辑结构，对所有逻辑路径进行测试。白盒测试法是穷举路径测试。在使用这一方法时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。贯穿程序的独立路径数是天文数字，但即使每条路径都测试了仍然可能有错误。第一，穷举路径测试不能查出程序违反了设计规范，即程序本身是个错误的程序。第二，穷举路径测试不可能查出程序中因遗漏路径而出现的错误。第三，穷举路径测试可能发现不了一些与数据相关的错误。

3. ALAC（Act-like-a-customer）测试

ALAC 测试是一种基于客户使用产品的知识开发出来的测试方法，基于复杂的软件产品有许多错误的原则。最大的受益者是用户，缺陷查找和改正将针对那些客户最容易遇到

的错误。

4. 单元测试

单元测试的对象是软件设计的最小单位——模块。单元测试的依据是详细设计描述，应对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。单元测试多采用白盒测试技术，系统内多个模块可以并行地进行测试。

5. 综合测试

时常有这样的情况发生，明明每个模块都能单独工作，但这些模块集成在一起之后却不能正常工作。主要原因是，模块相互调用时接口会引入许多新问题。例如，数据经过接口可能丢失；一个模块对另一个模块可能造成不应有的影响；几个子功能组合起来不能实现主功能；误差不断积累达到不可接受的程度；全局数据结构出现错误等。综合测试是组装软件的系统测试技术，按设计要求把通过单元测试的各个模块组装在一起之后，需进行综合测试以便发现与接口有关的各种错误。

6. 确认测试（集成测试）

通过综合测试之后，软件已完全组装起来，接口方面的错误也已排除，软件测试的最后一步——确认测试即可开始。确认测试应检查软件能否按合同要求进行工作，即是否满足软件需求说明书中的确认标准。

1）确认测试标准

实现软件确认要通过一系列黑盒测试。确认测试同样需要制订测试计划和过程，测试计划应规定测试的种类和测试进度，测试过程则定义一些特殊的测试用例，旨在说明软件与需求是否一致。无论是计划还是过程，都应该着重考虑软件是否满足合同规定的所有功能和性能，文档资料是否完整、准确，以及人机界面和其他方面（例如，可移植性、兼容性、错误恢复能力和可维护性等）是否令用户满意。

确认测试的结果有两种可能，一种是功能和性能指标满足软件需求说明的要求，用户可以接受；另一种是软件不满足软件需求说明的要求，用户无法接受。项目进行到这个阶段才发现严重错误和偏差一般很难在预定的工期内改正，因此必须与用户协商，寻求一个妥善解决问题的方法。

2）配置复审

确认测试的另一个重要环节是配置复审。复审的目的在于保证软件配置齐全、分类有序，并且包括软件维护所必需的细节。

7. α、β 测试

事实上，软件开发人员不可能完全预见用户实际使用程序的情况。例如，用户可能错误地理解命令，或提供一些奇怪的数据组合，也可能对设计者自认明了的输出信息迷惑不解，等等。因此，软件是否真正满足最终用户的要求，应由用户进行一系列验收测试才能肯定。验收测试既可以是非正式的测试，也可以是有计划的、系统的测试。有时，验收测试长达数周甚至数月，不断暴露错误，导致开发延期。一个软件产品，可能拥有众多用户，不可能由每

个用户验收，此时多采用称为 α、β 测试的方法，以期发现那些似乎只有最终用户才能发现的问题。

α 测试是指软件开发公司组织内部人员模拟各类用户对即将面市的软件产品(称为 α 版本)进行测试，试图发现错误并修正。α 测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作，并尽最大努力涵盖所有可能的用户操作方式。

经过 α 测试调整的软件产品称为 β 版本。紧随其后的 β 测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用 β 版本，并要求用户报告异常情况、提出批评意见。然后软件开发公司再对 β 版本进行改错和完善。

8. 系统测试

计算机软件是计算机系统的一个重要组成部分，软件开发完毕后应与系统中其他成分集成在一起，此时需要进行一系列系统集成和确认测试。对这些测试的详细讨论已超出软件测试的范围，这些测试也不可能仅由软件开发人员完成。在系统测试之前，软件工程师应完成下列工作。

(1) 为测试软件系统的输入信息设计出错处理。

(2) 设计测试用例，模拟错误数据和软件接口可能发生的错误，记录测试结果，为系统测试提供经验和帮助。

(3) 参与系统测试的规划和设计，保证软件测试的合理性。

系统测试应该由若干个不同测试组成，目的是充分运行系统，验证系统各部件是否都能正常工作并完成所赋予的任务。系统测试包括恢复测试、可靠性测试、安全测试、强度测试和性能测试等。

(1) 恢复测试：恢复测试主要检查系统的容错能力，即当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。恢复测试首先要采用各种办法强迫系统失败，然后验证系统是否能尽快恢复。对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性；对于人工干预的恢复系统，还需估测平均修复时间，确定其是否在可接受的范围内。

(2) 安全测试：安全测试检查系统是否具有对非法侵入的防范能力。安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。例如，想方设法截取或破译口令；专门定做软件破坏系统的保护机制；故意导致系统失败，企图趁恢复之机非法进入；试图通过浏览非保密数据推导所需信息，等等。理论上讲，只要有足够的时间和资源，没有不可进入的系统。因此系统安全设计的准则是，使非法侵入的代价超过被保护信息的价值，此时非法侵入者已无利可图。

(3) 强度测试：强度测试检查程序对异常情况的抵抗能力。强度测试总是迫使系统在异常的资源配置情况下运行。例如，当中断的正常频率为每秒一至两个，运行每秒产生十个中断的测试用例；定量地加大数据输入率，检查输入子功能的反应能力；运行需要最大存储空间(或其他资源)的测试用例；运行可能导致虚存操作系统崩溃或磁盘数据剧烈抖动的测试用例，等等。

(4) 性能测试：对于那些实时和嵌入式系统，软件部分即使满足功能要求，也未必能够满足性能要求，虽然从单元测试起，每一测试步骤都包含性能测试，但只有当系统真正集成

之后，在真实环境中才能全面、可靠地检查运行性能，系统性能测试是为了完成这一测试任务而诞生的。性能测试有时与强度测试相结合，经常需要其他软硬件的配套支持。

（5）可用性测试：即对“用户友好性”的测试。显然这是主观的，且取决于目标最终用户或客户。用户面谈、调查、用户对话的录像和其他一些技术都可使用。程序员和测试员通常都不宜作为可用性测试员。

（6）可靠性测试：可靠性测试是为了检验软件系统运行是否可靠而进行的一种测试。软件系统的失败往往导致不可预料的结果，如航空、航天领域中运行的软件，铁路系统中运行的软件等。可靠性测试的方法关心的是，一旦软件系统出现故障，其系统是否导向安全，所以可靠性测试与安全测试紧密相关。可靠性测试通常采用黑盒测试法。

9. 面向对象的软件测试

面向对象的软件测试(OO Testing)是针对根据面向对象软件开发方法设计的软件系统所提出的软件测试方法。OO Testing 又分为面向对象分析的测试(OOA Testing)、面向对象设计的测试(OOD Testing)和面向对象的程序测试(OOP Testing)。前两者是对分析结果和设计结果的测试，主要对分析设计产生的文本进行，是软件开发前期的关键性测试。OOP Testing 主要针对编程风格和程序代码实现进行测试，其主要的测试内容在面向对象单元测试和面向对象集成测试中体现。面向对象单元测试是对程序内部具体单一的功能模块的测试，如果程序是用 C++语言实现，主要就是对类成员函数的测试。面向对象单元测试是进行面向对象集成测试的基础。面向对象集成测试主要对系统内部的相互服务进行测试，如成员函数间的相互作用，类间的消息传递等。面向对象集成测试不但要基于面向对象单元测试，更要参见 OOD 或 OOD Testing 结果。面向对象系统测试是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准，需要借鉴 OOA 或 OOA Testing 结果。

10. 协议软件测试

在计算机网络的发展历程中，协议一直处于核心地位，它是计算机网络和分布式系统中各种通信实体之间相互交换信息所必须遵守的一组规则。1984 年，国际标准化组织 ISO 提出了开放式系统互联 ISO/OSI 参考模型。1993 年 1 月 1 日，TCP/IP 被宣布为 Internet 上唯一正式的协议，为 Internet 的发展铺平了道路。

协议软件作为软件的一种特殊形式，自 20 世纪 80 年代以来，其开发和检测方法已经得到快速的发展，已经形成了一个崭新的学科——协议工程学，它的研究范围包括：协议说明(protocol specification)、协议证实(protocol validation)、协议验证(protocol verification)、协议综合(protocol synthesis)、协议转换(protocol conversion)、协议性能分析(protocol performance analysis)、协议自动实现(protocol automatic implementation)和协议测试(protocol testing)。

目前的网络协议多是以自然语言描述的文本，实现者对于协议文本的不同理解以及实现过程中的非形式化因素都会导致不同的协议实现，有时甚至是错误的协议实现。即便协议实现正确，也不能保证不同的实现彼此之间能够准确无误地通信，而且同一协议的不同实现其性能也有差别。在这种情况下，需要一种有效的方法对协议实现进行评价，这就是“协

议测试”。

目前的协议测试已经不仅仅是产品开发研制过程中一个简单的检测支持过程，而发展成为计算机网络技术的一个重要分支。对协议测试技术的研究将直接影响到计算机网络技术的进步和世界网络市场的竞争与发展。所以很多国家都投入了大量的人力物力从事协议测试的研究工作。例如，英国的国家物理实验室（NPL）、法国国家通信研究中心、德国国家通信研究局（GMD）、美国国家标准化研究局、美国新罕布什尔大学互操作研究实验室、中国清华大学计算机科学与技术系的计算机网络与协议测试实验室等单位都在这个领域投入了大量的研究力量。

协议测试是在软件测试的基础上发展起来的。协议测试是一种黑盒测试，它按照协议标准，通过控制观察被测协议实现的外部行为对其进行评价。目前协议测试分成三个方面进行研究：一致性测试（conformance testing）、互操作性测试（interoperability testing）和性能测试（performance testing）。一致性测试主要测试协议实现是否严格遵循相应的协议描述；互操作性测试关注的是对于同一个协议标准，不同协议实现之间的互连通问题；性能测试是用实验的方法来观测被测协议实现的各种性能参数，如吞吐量和传输延迟等，其结果往往与输入负载有关。

在上述三个方面，一致性测试开展最早，也形成了很多有价值的成果。1991 年，国际标准化组织 ISO 制定的国际标准 ISO 9646——《OSI 协议一致性测试的方法和框架》，用自然语言描述了基于 OSI 七层参考模型的协议测试过程、概念和方法。但是随着计算机网络技术的不断发展，新的协议越来越复杂，协议一致性测试工作遇到了很多困难。在这个过程中，大量形式化方法被引进到协议测试研究领域。1995 年，ISO 推出了“一致性测试中的形式化方法”国际标准，对协议一致性测试过程各个阶段使用的形式化方法进行了说明。但由于协议一致性测试本身的复杂性，使得该标准一直停留在草案阶段。对于互操作测试的研究技术基本上是从一致性测试继承过来的。由于对网络应用的需求急剧增长，网络性能已经变得与功能同等重要了。协议实现性能测试的研究工作也正在进行之中。在进行大量的测试实践的同时，理论研究也正在起步。

目前，国际协议测试研究领域已经取得了以下两点共识：

第一是理顺了协议一致性测试的过程。

第二是将形式化技术引入了协议测试领域，力求用严格的数学语言清晰、无二义性地研究协议测试的概念和方法。

但是这种方法也存在着很多不足，其中最明显的就是理论研究与实际应用之间还存在着巨大的差距。

1.4.5 软件测试人员的基本素质

现在许多软件企业招收一些刚刚毕业的大学生或者非计算机专业的人员作为自己公司软件测试工程师，这是非常错误的，也是对软件测试不负责任的表现。虽然他们可以发现软件中的一些错误，但是对于软件中的一些关键、致命、危险的错误他们是很难发现的。大家都知道，软件工程中有个模型叫瀑布模型，这是最基本的软件模型，这个模型又叫碗状模型，因为开发位于碗的最底部，左上方依次为建模、需求分析、设计，右上方依次为测试、部署、维护。这就说明软件开发是一切软件活动的基础，同时也是软件测试的基础。一个人只有经

历过一定年限的软件开发工作，才可以积累丰富的经验，知道在软件中哪些地方容易出错而哪些地方不容易出错，这会给以后的软件测试工作带来非常宝贵的经验。概括地说，软件测试人员应该具备以下基本素质。

1. 有逆向思维的能力

一些软件测试工程师，他们干了一段时间软件测试工作后返回去又开始去做开发工作了，这是为什么呢？答案是软件测试工作太难了，开发是顺向思维，而测试是逆向思维，总要找一些稀奇古怪的思路去操作软件。软件的使用者千差万别，软件在使用过程中遇到的各种现象也是千差万别的，所以要求软件测试工程师需要具有一些逆向思维的能力，想别人所不想，测别人所不测，这样才可以找到更多的软件中的错误。这是作为一名优秀的软件测试工程师最基本的素质。

2. 善于同软件开发人员沟通

沟通是当今软件项目中需要掌握的最关键技术之一。软件测试人员要善于同软件开发人员沟通，软件测试人员与开发人员搞好关系，使测试人员不成为开发人员的眼中钉，这对于提高整个软件项目质量是十分重要的。

3. 讨论软件的需求、设计

通过这样的讨论，你可以更好地了解所测试的软件系统，以至于尽可能少地测试出软件中不是错误的“错误”，从而降低给软件开发人员带来的压力。

4. 报告好的测试结果

作为一个测试人员，发现错误往往是测试人员最愿意而且引以为自豪的结果，但是一味地给开发人员报告软件错误，会给他们造成厌恶感，降低整个软件的质量和开发进度。所以作为一名软件测试工程师，当你测试的模块没有严重的错误或者错误很少的时候，你不妨跑到开发人员那里告诉他们这个好消息，这会给你带来意想不到的结果。

5. 讨论一些与工作无关的事情

作为一个测试人员，经常和开发人员讨论一些与工作无关的事情，比如大家可以谈谈新闻、趣事、家庭等，这样可以加强相互间的默契程度。许多统计表明，这样可以更好地提高软件工作质量。

6. 善于同领导沟通

测试人员往往是领导的眼和耳，领导根据测试人员的测试结果可以了解公司的产品质量，从而调整其他工作。领导工作一般比较繁忙，所以作为一名优秀的测试人员要学会把测试结果进行总结，最好以图表的形式给领导看。

7. 掌握一些自动化测试工具

测试往往是比较烦琐、枯燥无味的工作，测试人员长期处于单调的手工重复，会降低测

试效率,并且对于测试质量也往往是不利的;况且许多测试不使用测试工具是不可以进行的,比如性能测试、压力测试等。目前市场上有许多测试工具供你使用,你可以根据自己的需要选择一些测试工具来辅助你的测试。但是要记住一点,不是说有了测试工具就不要人工测试了,测试工具不是万能的。

8. 善于学习的能力

软件测试技术随着时间的变化也在做一些提高和改进,作为一名优秀的测试人员要善于利用书籍、网站、论坛、交流等各种途径不断提高自己的软件测试水平。

9. 了解业务知识

了解测试软件的业务知识是非常重要的,对业务知识了解得越深入,越能够找出更深入、更关键、更隐蔽的软件错误。所以作为一名优秀的软件测试工程师,要多向该领域专家、同行学习,提高自己的业务知识水平。

10. 提高自己的表达能力

软件测试人员当发现软件中存在缺陷的时候,往往要书写缺陷报告,缺陷报告要写得详尽清楚,使开发人员能够尽快定位错误,修改错误,所以作为一名优秀的测试人员提高自己的写作能力是非常必要的。

1.5 本章小结

本章对软件测试的基础知识进行了简要的介绍,这些内容涵盖软件、软件危机、软件工程、软件缺陷和软件故障的相关概念,软件质量与质量模型,软件测试概要,以及要成为一名合格软件测试人员应该具备的基本素质。通过本章的学习可以对软件测试有一定的认识,为以后章节的进一步学习打下基础。

习题 1

1. 简述软件危机。
2. 说明软件缺陷、软件错误和软件失败三者的关系。
3. 简述外部度量、内部度量和使用度量。
4. 黑盒测试是一种穷举测试,试说明穷举测试的含义。
5. 简述软件测试人员应具备的一些素质。

第2章 软件测试计划

了解软件测试的基本概念后，就要开始通过一系列的测试活动来完成各个测试阶段所规定的任务。一般来讲，应由一位对整个系统设计熟悉的设计人员编写测试大纲，明确测试的内容和测试通过的准则，设计完整合理的测试用例，从而在系统实现后快速开展全面测试。虽然每个阶段具体的任务和要求不一样，但软件测试工作的基本范畴是一样的。本章将讨论测试工作开始前一项非常重要的工作——测试计划。专业的测试必须以一个好的测试计划作为基础，测试计划应该作为测试的起始步骤和重要环节。

根据《ANSI/IEEE软件测试文档标准829—1983》定义，软件测试计划是一个叙述了预定的测试活动的范围、途径、资源及进度安排的文档，它确认了测试项、被测特征、测试任务、人员安排，以及任何偶发事件的风险。

软件测试计划是指导测试过程的纲领性文件，包含产品概述、测试策略、测试方法、测试区域、测试配置、测试周期、测试资源、测试交流、风险分析等内容。

本章的主要内容如下：

- 什么是测试计划？
- 测试计划的内容。
- 如何做好测试计划？

2.1 软件测试计划的作用

软件测试计划作为软件项目计划的子计划，在项目启动初期是必须规划的。在越来越多公司的软件开发中，软件质量日益受到重视，测试过程也从一个相对独立的步骤越来越紧密地嵌套在软件整个生命周期中，这样，如何规划整个项目周期的测试工作，如何将测试工作上升到测试管理的高度都依赖于测试计划的制定。测试计划因此也成为测试工作开展的基础。

一个好的测试计划有助于测试任务的完成。为了创建一个好的测试计划，必须以一种系统的方式对程序进行调查，使得对程序的处理更清晰、更彻底、更有效。

(1) 提高测试覆盖率。在做测试计划时，要求有一个程序测试清单。有了这样的测试清单，在测试中就不会遗漏任何一项测试。必须找出所有与测试相关的内容。通常有效的做法是：在清单中列出由程序创建的所有报告以及所有错误信息、所有支持的打印机、所有菜单选项、所有对话框、每一对话框中的所有选项等。清单内容越详细，因不了解而遗漏的

东西就越少。

(2) 避免不必要的重复和遗忘项目。核对测试清单或图表上所列的项目时,很容易就能够看出哪些内容测试过,哪些内容没测试过。

(3) 分析程序,快速选择出合适的测试用例。在测试文档中,对等价类和边界条件的数据录入字段进行分析,得出的每一个边界都是一个很好的测试用例,因为边界值要比非边界值更可能发现缺陷。

(4) 提供测试结果。所有的编码工作完成后,测试就开始了,产品发布之前,测试人员的压力很大,而且常常只有很少的时间可以用来安排最终测试。这时以前的测试文档将帮助你确保最后一次运行重要测试。如果没有这些文档,单凭记忆是很难记住哪些测试是需要重新运行的。

(5) 减少测试数量,但不增加所遗漏的缺陷数量,可以提高测试效率。主要的方法就是从那些类似的测试用例中挑选其中的一部分,而不是所有。因为运行同一类型的测试用例所得到的结果也是类似的。

(6) 检查测试的完整性。如果不能确定程序的某一部分是否进行过某一项测试,就可以对照清单来检查一下,即检查测试的完整性。

在制定测试计划过程中,可以和团队中的其他人员多进行交流,这种良好的沟通、交流有以下几个优点:

(1) 测试人员可以交流制定测试策略的思想。

(2) 测试人员可以得到测试准确度和覆盖率的反馈。通过这些反馈,测试人员可以发现遗漏的尚未测试的程序区域,以及对程序的一些误解。

(3) 有助于测试人员了解测试工作的规模。测试计划内容包括所要进行的具体工作以及已完成工作的数量,这会帮助经理及其他人了解完成测试工作需要花费多少时间。那么,项目经理就可以根据项目的实际情况考虑简化或淘汰某项测试。

(4) 有助于顺利进行工作分派。如果能给后续环节的测试人员提供一个详细的书面指令集,那么委派或监督产品的测试就要容易得多。

2.2 软件测试计划的原则

制定测试计划是软件测试中最有挑战性的一个工作,以下原则将有助于制定测试工作计划:

- 制定测试计划应尽早开始。
- 保持测试计划的灵活性。
- 保持测试计划简洁和易读。
- 尽量争取多渠道评审测试计划。
- 计算测试计划的投入。

除了上面讲的原则外,要做好测试计划,还必须注意以下几点。

1. 明确测试的目标,增强测试计划的实用性

测试目标必须是明确的,可以量化和度量的,而不是模棱两可的宏观描述。另外,测试

目标应该相对集中，避免一股脑罗列出一系列目标，造成轻重不分或平均用力。应根据对用户需求文档和设计规格文档的分析，确定被测软件的质量要求和测试需要达到的目标。

2. 坚持“5W1H”规则

所谓“5W1H”规则即：

Why：为什么要进行这些测试。

What：测试哪些方面，不同阶段的工作内容。

When：测试不同阶段的起止时间。

Where：相应文档，缺陷的存放位置，测试环境等。

Who：项目有关人员组成，安排哪些测试人员进行测试。

How：如何去做，使用哪些测试工具以及测试方法进行测试。

3. 采用评审和更新机制，保证测试计划满足实际需求

测试计划写完后，如果没有经过评审，就直接发送给测试团队，难免发生内容不准确或遗漏测试内容，或者软件需求变更引起测试范围的增减，而测试计划的内容没有及时更新，从而误导测试执行人员。测试计划包含多方面的内容，而编写人员可能自身测试经验和对软件需求的理解都有限，而且软件开发是一个渐进的过程，所以最初创建的测试计划可能是不完善的、需要更新的。需要采取相应的评审机制对测试计划的完整性、正确性、可行性进行评估。例如，在创建完测试计划后，提交到由项目经理、开发经理、测试经理、市场经理等组成的评审委员会审阅，根据审阅意见和建议进行修改和更新。

4. 分别创建测试计划与测试详细规格、测试用例

编写软件测试计划要避免一种不良倾向，就是测试计划的“大而全”，篇幅冗长，重点不突出，既浪费写作时间，也浪费测试人员的阅读时间。最好的方法是把详细的测试技术指标包含到独立创建的测试详细规格文档，把用于指导测试小组执行测试过程的测试用例放到独立创建的测试用例文档或测试用例管理数据库中。测试计划和测试详细规格、测试用例之间是战略和战术的关系，测试计划主要从宏观上规划测试活动的范围、方法和资源配置，而测试详细规格、测试用例是完成测试任务的具体战术。

5. 测试阶段的划分

就通常的软件项目而言，基本上采用“瀑布型”开发方式，这种开发方式下，各个项目主要活动比较清晰，易于操作。整个项目生命周期为“需求—设计—编码—测试—发布—实施—维护”。然而，在制定测试计划时，有些测试经理对测试的阶段划分还不是十分明晰，经常遇到的问题是把测试单纯理解成系统测试，或者把各类型测试设计全部放入生命周期的“测试阶段”，这样一方面浪费了开发阶段可以并行的项目日程，另一方面也造成测试不足。

6. 系统测试阶段日程安排

划分阶段清楚了，随之而来的问题是测试执行需要多长的时间。标准的工程方法是对工作量进行估算，而后得出具体的估算值，但是这种方法过于复杂，可以另辟专题讨论。一

个可操作的简单方法是：根据测试执行上一阶段的活动时间进行换算，换算方法是与上一阶段活动时间比较，比值维持在 1∶1.1～1.5。也就是说开发计划中的系统测试的时间大概是编码阶段(包含单元测试和集成测试)的 1～1.5 倍。这种方法的优点是简单，依赖于项目计划的日程安排，缺点是水分太多，难于量化。

2.3 如何制定软件测试计划

在制定测试计划时，需要经历如图 2-1 所示的几个步骤。

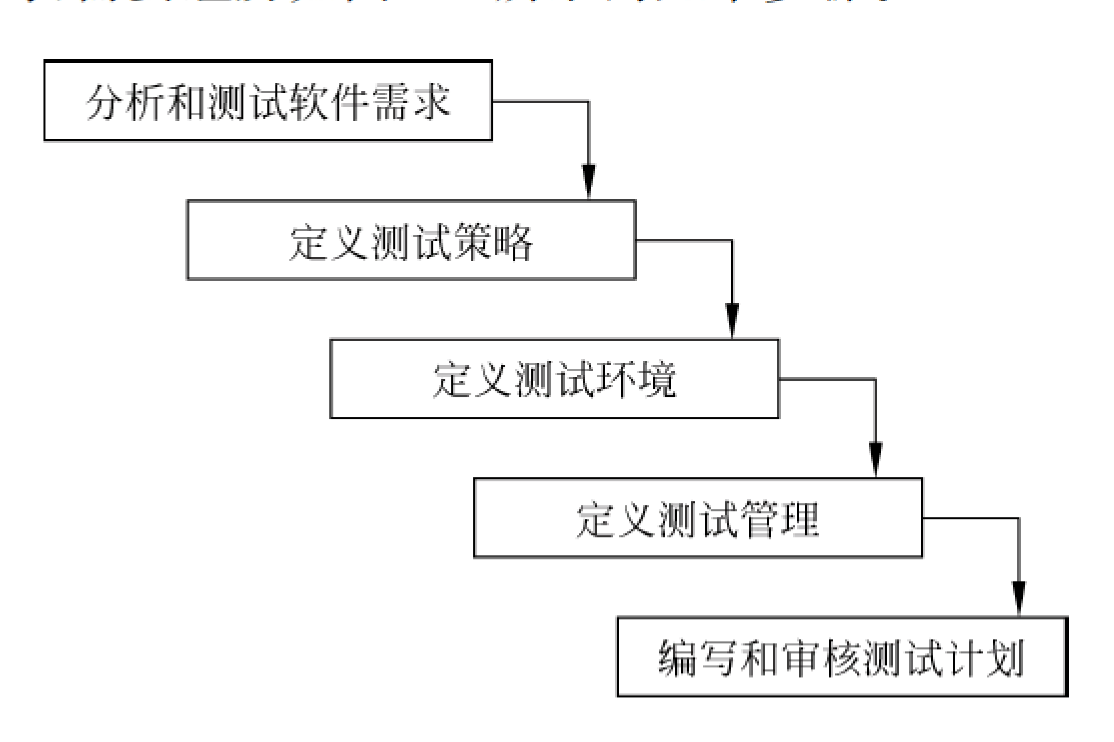


图 2-1 测试计划制定过程

软件测试计划是指导测试过程的纲领性文件，它描述了测试活动的范围、方法、策略、资源、任务安排和进度等，并确定测试项(哪些功能特性将被测试，哪些功能特性将无须测试)，识别测试过程中的风险。借助软件测试计划，可确保测试实施过程顺畅，有效地跟踪和控制测试过程，并应对可能发生的各种变更。

在制定测试计划时，由于不同软件公司的背景不同，测试计划内容会有差异，但一些基本内容是相同的。例如，IEEE 829—1998 软件测试文档编制标准中规定软件测试计划应包含 16 项内容：

- 测试计划标识符。
- 项目总体情况简介。
- 测试项(test item)。
- 需要测试的功能。
- 方法策略。
- 不需要测试的功能。
- 测试项通过/失败的标准。
- 测试中断和恢复的规定。
- 测试完成所提交的材料。
- 测试任务。
- 测试环境要求。
- 测试人员职责。
- 人员安排与培训需要。

- 进度表。
- 潜在的问题和风险。
- 审批。

在测试计划中，还要考虑休假和法定假日带来的影响，以及做好项目相关技术和业务的培训。制定软件测试计划主要集中在测试目标和需求说明、测试工作量估算、测试策略、测试资源配置、进度表、测试风险等。

- 目标和范围。包括质量目标、产品特性、各阶段的测试对象、目标、范围和限制。
- 项目估算。根据历史数据和采用恰当的评估技术，如项目工作分解结构方法，对测试工作量、所需资源做出合理的估算。
- 风险计划。测试可能存在的风险的分析、识别，以及风险的回避、监控和管理。
- 进度表。根据项目估算结果和人力资源现状，以软件测试的常规周期作为参考，采用关键路径法等，完成进度的安排，采用时限图、甘特图等方法来描述资源和时间的关系，例如什么时候测试哪一个模块、什么时候要完成某项测试任务等。
- 项目资源。人员、硬件和软件等资源的组织和分配，人力资源是重点，它和日程安排联系密切。
- 跟踪和控制机制。质量保证和控制，变化管理和控制等。例如，明确如何提交一个问题报告、如何去界定一个问题的性质或严重程度、多少时间内做出响应等。

对于大型软件项目，可能需要一系列测试计划书，例如按集成测试、系统测试、验收测试等阶段去组织，为每一个阶段制定一个计划书，也可以为每个测试项如安全性测试、性能测试、可靠性测试等制定特别的计划书，甚至可以制定测试范围/风险分析报告、测试标准工作计划、资源和培训计划、风险管理计划、测试实施计划、质量保证计划等。

2.4　制定测试计划时面对的问题

制定测试计划时，测试人员可能面对以下问题，必须认真对待，并妥善处理。

1. 与开发者意见不一致

开发者与测试者在测试工作上经常处于对立的立场，双方都认为对方一心想要占上风。通常，这种心态只会牵制项目，耗费精力，还会影响双方的关系，而不会对测试工作起任何积极作用。

2. 缺乏测试工具

项目管理部门可能对测试工具的重要性缺乏足够的认识，导致人工测试通常占了绝大多数。

3. 培训不够

相当多的测试人员没有接受过正规的测试培训，这会导致对测试计划大量的误解和无用测试。

4. 管理部门缺乏对测试工作的理解和支持

对测试工作的支持必须来源于上层。这种支持不仅仅是资金投入，还应该对测试工作遇到的问题给出一个明确的态度，否则，测试人员的积极性将会受到影响。

5. 缺乏用户的参与

用户可能被排除在测试工作之外，或者可能是他们自己不想参与进来。事实上，用户在测试工作中的作用相当重要，他们能够确保软件适用于实际应用。

6. 测试时间不足

这是一种普遍的抱怨。问题在于如何将计划各部分划分出优先级，以便在给定的时间内测试应该测试的内容。

7. 过分依赖测试人员

项目开发人员知道测试人员会检查他们的工作，所以他们只集中精力编写代码，而把问题留给测试人员测试发现。这样通常会导致更高的缺陷级别和更长的测试时间。

8. 测试人员处于进退两难的状态

一方面，如果测试人员报告了太多的缺陷，那么大家会责备他们延误了项目；另一方面，如果测试人员没有找到关键性的缺陷，大家又会责备他们的工作质量不高。

9. 不得不说“不”

对于测试人员来说这是最尴尬的境地，有时不得不说“不”。项目相关的人员都不愿意听到这个“不”字，而且通常测试人员要屈从于进度和费用的压力。

2.5 衡量测试计划的标准

一份好的测试计划书应具备哪些特点呢？总结起来，应具备以下几点：

(1) 它应能有效地引导整个软件测试工作正常运行，并配合编程部门，保证软件质量，按时将产品推出。

(2) 它所提供的方法应能使测试高效地进行，即能在较短的时间内找出尽可能多的软件缺陷。

(3) 它提供了明确的测试目标、测试策略、具体步骤及测试标准。

(4) 它既强调测试重点，也重视测试的基本覆盖率。

(5) 它所制定的测试方案尽可能充分利用了公司现有的、可以提供给测试部门的人力/物力资源，而且是可行的。

(6) 它所列举的所有数据都必须是准确的，例如，外部软件、硬件的兼容性所要求的数据，输入、输出数据等。

(7) 它对测试工作的安排有一定的灵活性，可以应付一些突然的变化情况，如当时间安

排或产品出现一些变化的时候，测试工作也相应变化。

2.6 制定测试计划

在完成了测试软件需求分析之后，就可着手制定策略。当测试组长开始做测试计划时，需要首先考虑以下问题。

1. 测试的范围

由于软件是无法被完全测试的，因此对于被测试软件，要判断哪些功能、特性需要被测试。

2. 测试的方法

对于不同系统，需要采用不同的测试方法，另外，在有些时候，可能并不进行某些类型的测试。

3. 质量标准

在开发的每个阶段，都需要对该阶段完成的软件版本定义质量标准，不同阶段的版本的质量标准是不一样的。例如，给用户进行演示以获取更多反馈的版本和最后递交给客户的版本，其质量要求肯定是不一样的。另外，测试本身也具有一个规范的流程和循序渐进的过程，只有完成了一个阶段的测试，才能开始下一个阶段的测试，这样，就需要对每个阶段在什么情况下可以开始、什么情况下可以结束进行定义。

4. 自动化测试工具的选择

在进行一些项目的测试时，需要使用自动化测试工具。在制定测试策略时，需要判断是否使用自动化测试工具、使用什么自动化测试工具。

5. 测试软件的编写

测试软件包括以下几种类型：自动化测试软件、仿真软件和运行环境软件。在进行一些项目的测试时，需要组织编写这些软件。

6. 与项目相关的一些特殊的考虑

在很多项目中，会面临一些相关的特殊情况。例如，由于项目进度很紧张，导致需要程序员在白天工作，测试员在晚上工作。

测试策略一般按照上面的步骤进行定义，但在很多时候，这些步骤可能是并行进行的。例如，考虑到部分测试可能需要自行编写测试软件，在考虑测试方法的时候这个软件的开发工作可能就会开始。

2.6.1 确定测试范围

在进行软件测试之前，需要分析产品的需求文档来决定哪些功能需要被测试，而且一些

文档之外的过程也必须考虑，如产品的安装、升级测试，可用性测试，在客户环境中和其他设备的协同性测试。

完全覆盖的软件测试是不可能实现的，所以需要确定测试的范围。例如 Windows 附件中的计算器软件，如果要完全测试其中两个整数相加的功能，就需要把所有可能的组合都测试一次，而这个组合有 $2\times10^{32}\times2\times10^{32}$ 个。对于这么庞大的一个测试量，实际是不可能完全实现的。而两个整数相加仅仅是计算器中的一个小功能，简单的计算器软件尚且如此，更何况一些比较复杂的软件系统，想要对其进行完全测试，是根本不可能实现的。

还存在这样一些比较典型的情况，使得我们可以事先确定一个必需的测试范围而不是测试所有的内容：

(1) 某些阶段的测试或者某些内容的测试可以简化。有时，软件是在旧版本的基础上进行新版本的开发的，在旧版本中，有些模块已经进行过很多次单元测试，证明这个模块是足够坚固的，在新版本的开发中就不需要对这个模块再进行单元测试。

(2) 当对原有系统进行修改升级时，某些测试不需要。例如为某个软件添加打印预览功能，这时文件打开和保存的功能就不需要进行很多测试。当然在这种情况下，需要慎重选择哪些部分会受到新功能的影响，需要测试，哪些不会受到影响，不需要过多的测试。

(3) 某些测试根本不可能进行。例如，游戏公司开发了某个游戏的新的功能，根本不可能让所有的游戏用户在某个时间段去试用，而只能采取一些模拟的手段进行测试。

由于不可能测试所有内容，因此决定要测试什么变得非常重要。如果测试过度，意味着在测试覆盖中存在大量冗余，会花费大量时间，如果测试范围过小，就存在遗漏 Bug 的风险。确定测试范围就是在测试时间、测试费用和质量风险之间寻找平衡的过程，期望花费更少的时间和费用，就必然承担更大的质量风险，找到两者之间的平衡，需要经验以及一个可以评价测试成功的标准。

在确定测试范围时，可以结合需要被测试的软件，考虑下列一些因素：

- 测试最高优先级的需求。假如需求分析中对用户的需求的优先级作了定义，选择对用户最重要的需求进行测试。
- 测试新功能代码或者改进旧功能。对一个产品的初始版本，所有的内容都是新的，但对于产品的升级或维护版本，测试可集中在新的代码上。在实际操作过程中，改动过的代码有时也会影响那些没有改动过的代码的运行，所以代码改动后尽可能地进行回归测试来测试所有的程序功能。
- 使用等价类划分来减小测试范围。例如上面提到的计算器软件，可以认为 2 个三位的正数相加是一个等价类。
- 重点测试经常出现问题的地方。在软件中，如果某个代码模块、功能模块出现过较多的问题，那么，它就有可能还有更多的问题。

为了有效缩小测试范围，可以建立一份提问单，基于这份提问单，可以找到最需要测试的内容。表 2-1 就是一份简单的测试提问单。

测试范围的确定不仅仅是由测试人员决定的，需求分析人员、设计人员、程序员、市场/销售人员、公司工程管理人员、客户都有可能参加这个过程。在测试的过程中，测试的范围也不一定是一成不变的。例如客户突然要求提前使用产品，这时就必须调整测试范围。

表 2-1 测试提问单

问　　题	回　　答
哪些功能是软件的特色?	
哪些功能是用户最常用的?	
如果系统可以分块销售,那么哪些功能块在销售时价钱最昂贵?	
哪些功能出错将导致用户不满或索赔?	
哪些程序是最复杂、最容易出错的?	
哪些程序是相对独立,应当提前测试的?	
哪些程序最容易扩散错误?	
哪些程序是全系统的性能瓶颈所在?	
哪些程序是开发者最没有信心的?	

2.6.2 选择测试方法

在不同的开发阶段,需要采用不同的测试方法。以瀑布式生命周期模型为例,在不同阶段可选择不同的测试方法。

(1) 需求分析阶段。被测试的对象主要是需求文档,这时可以用静态的方式进行测试。

(2) 概要设计与详细设计阶段。需要完成结构设计和详细设计文档,与需求分析阶段类似,也是用静态的方式进行测试。只是在实际软件开发过程中,测试人员往往不对概要设计和详细设计进行测试。

(3) 编码和单元测试阶段。在编码阶段,往往是采用一些诸如代码走查这样的静态测试方式,或利用 JUnit 等单元测试工具进行动态白盒测试。

(4) 集成测试阶段。主要采用一些动态的测试技术。

(5) 系统测试阶段。和集成测试阶段类似,也是采用一些动态测试技术和黑盒测试方法。但在本阶段,重点会放在压力测试、负载测试、安全测试、升级测试、可用性测试等方面。

(6) 验收测试阶段。一般需要用户加入本阶段的过程,有时甚至完全由用户进行测试。该阶段的测试完全采用动态测试和黑盒测试技术。

在哪个阶段选择哪种测试方法,并没有严格规定,需要测试人员根据项目实际情况来决定。在选择测试方法时,参加到这项工作的测试人员应具备各种测试方法的知识、了解软件需求、了解该软件所需要达到的质量标准。

2.6.3 测试标准

在选择好测试方法后,就需要确定测试标准。软件测试是按照事先被定义的流程来进行的,从一个步骤进入到下一个步骤,需要根据一定的标准判断是否可以进入下一个环节。定义测试标准的目的是制定测试过程中需要遵守的规则,测试标准实际上也是软件的质量标准之一。

在制定测试计划时就需要着手制定测试标准,如果测试工作开始时没有确定测试标准,测试时就没有良好的评价依据。例如,要求开发人员必须先完成单元测试,并且在单元测试后系统必须达到一定的质量标准,才能递交测试人员进行集成测试和系统测试。

在测试计划阶段，一般需要明确的测试标准有测试入口标准、测试暂停与继续标准、测试出口标准。

1. 测试入口标准

测试入口标准用以说明在进行某个阶段的测试时，需要具备什么样的前提条件。如果上午刚编写完的系统，并没有进行单元测试，直接做集成测试是不现实的，只会无端消耗测试人员的时间。

不同的公司、不同的项目、不同的测试阶段制定出来的入口标准是不同的，需要测试人员根据实际情况，结合项目组其他成员定制入口标准，这些标准往往包括需要准备的文档和软件需要达到的质量。以系统测试阶段为例，在开始进行系统测试之前，需要完成的工作有：完整的软件包，包括软件的安装光盘和相应的手册；系统测试计划和所使用的测试案例；测试数据；所需的测试环境；软件已经通过集成测试。在实际的项目中，会根据以上几点进行详细的定义，这些定义就形成了系统测试的入口标准。

2. 测试暂停与继续标准

在测试过程中有可能因为一些因素而意外暂停，在什么情况下需要测试工作暂时停止就是暂停标准。暂停的发生往往是由于软件的质量问题导致的，例如在进行集成测试时，可以建立这样的标准：当测试人员在一个测试工作日发现 Bug 的数量超过 50 个时，停止集成测试，由开发人员重新进行单元测试。

另外，有很多其他因素也会导致测试工作的暂停，如测试环境未能准备好、测试工具未能准备好、未能完成人员培训、发现缺陷数量过少等。发现缺陷数量很多，这往往是编码的质量太差而导致的；发现缺陷数量过少，有可能是编码质量很高，也有可能由于测试用例不完善导致不能发现代码中的错误，或由于测试人员不熟悉被测软件所处的行业领域问题而不能发现代码中的错误。这时，需要测试人员暂时停止测试，寻找问题发生的原因。

测试继续标准和测试暂停标准是对应的，在测试暂停标准中，列出了在什么情况下，测试将会暂停，而测试继续标准保证在问题被解决或确认有方法解决之后，测试可以继续进行。

3. 测试出口标准

测试出口标准用于说明在什么情况下可以结束某个阶段的测试。与测试入口标准相似，不同的公司、不同的项目、不同的测试阶段所制定的出口标准是不同的。以系统测试阶段为例，测试的入口标准定义了系统测试计划和所使用的测试案例，在测试的出口标准中就会定义所有测试案例都应被执行，并且未能通过的测试案例应该小于某个数值。值得注意的是，一个阶段的出口标准和另一个阶段的出口标准是不同的，要想进入下一个阶段进行测试，首先要达到上一个阶段的测试出口标准，并给下一个阶段的测试做必要的准备。例如并不是完成集成测试就可以开始系统测试，除了完成集成测试外，还需要准备审查系统测试的测试计划、测试案例等，然后才可以开始进行系统测试。

零缺陷的软件是不存在的，只要进行测试，就会不断地发现问题。显然，测试不可能一直进行下去，所以当软件达到一定的质量标准后，就可以通过该阶段的测试。那么，某个阶

段的测试什么时候可以结束呢？以下是三种比较实用的规则。

（1）基于测试用例的规则

基于测试用例的规则首先应该构造测试用例，如果测试用例的不通过率达到20%，则停止测试，待开发人员修正软件后再进行测试。如果软件的功能测试用例的通过率达到100%，非功能性测试用例的通过率达到90%时，允许正常结束测试。该规则的优点是适用于所有的测试阶段，缺点是太依赖测试用例，如果测试用例设计得非常差，该规则的作用就不大了。

（2）基于"测试期缺陷密度"的规则

如果把测试每小时发现的缺陷数称为"测试期缺陷密度"，就可以绘制出"测试时间-缺陷数"的关系图，缺陷达到一定密度停止测试。

（3）基于"运行期缺陷密度"的规则

把软件运行每小时发现的缺陷数称为"运行期缺陷密度"，可以绘制出"运行时间-缺陷数"的关系图，在相邻的N小时"运行期缺陷密度"都小于某个值M时，允许正常结束测试，该规则适用于验收测试阶段。

系统测试虽然是测试的最后一个阶段，但系统测试的出口标准并不是软件的发布标准，实际的项目开发中，除了要完成系统测试外，还需要有一些必要的工作才能进行软件发布，而且软件的发布是由项目经理或公司高层来决定的，软件达到系统测试出口标准，只是其中一个参考因素。

一般的软件产品都有其发布的质量标准，这个标准往往不是"零错误"，而是规定：灾难级Bug不允许存在；严重级Bug不允许存在；重要级Bug不允许存在；一般级Bug数量小于5；次要Bug数量小于10。在为用户开发系统所签的合约中，对存在Bug的数量也有明确的约束。

2.6.4　自动化测试工具的选择

在测试过程中，单纯使用手工测试几乎不能完成测试工作，使用自动化测试工具可以大大提高测试效率。在制定测试计划时，就应明确是否使用自动化测试工具，如果使用，在哪个阶段使用什么样的测试工具。

使用测试工具的优点有以下四个方面。

1. 能够很好地进行性能测试和压力测试

很多软件在进行性能测试和压力测试时，单纯的手工测试往往不能完成任务。

2. 能够改进回归测试

当测试人员测试完系统，将软件的测试报告发送编程人员后，编程人员将会根据测试报告中所指出的错误进行修改，完成系统错误的修复工作再次进行测试时，首先应执行相应的测试用例，验证以前发现的错误是否已被修复，那么其他的测试用例是否需要被运行呢？答案是需要的，这就是回归测试，之所以要这样做，主要是因为编程人员在修复被发现的错误时，很可能会制造新的错误。回归测试需要大量的时间，所以，测试人员并不能对修改后的每个新版本都进行回归测试，但在自动化测试工具的帮助下，迅速地执行测试用例，从而执

行更多的回归测试。

3. 能够缩短测试周期

自动化测试工具执行一个测试用例所耗费的时间，要比人工操作减少很多。只是编写能够被自动化测试工具识别的测试用例、测试脚本也是需要时间的。

4. 能够提高测试工作的可重复性

测试工作本身也可能会发生错误，测试人员在进行人工测试时，每次不一定使用同样的方式、同样的步骤来操作，而自动化测试工具可以保证在每次测试的时候都是完全按照同样的方式进行的。有些测试人员在使用自动化测试工具时往往会认为——自动化测试工具是无所不能的。因此在测试计划阶段，会把过多的精力投入到选择、学习使用自动化测试工具，最终反而没有找到合适的测试工具。过多地依赖自动化测试工具，反而会使测试工具的优势不能很好地发挥出来，影响测试工作的进行和测试的质量。

2.6.5 测试软件的编写

如果选择不到合适的自动化测试工具，就需要人工编写相应的测试软件。例如某个表格控件，对其进行测试就比较烦琐，需要用不同的语言来编写测试软件，如 VB、VC++、C++ Builder，不容易找到合适的自动化测试工具。需要假设编程流程和习惯，在这个基础上编写测试软件。

如果软件中某个部分需要自行编写测试软件来完成测试，就需要在测试计划阶段对此进行规划。测试软件有时由程序员来编写，有时也可以由测试人员来编写，这要根据软件开发公司的人员技术能力状况来决定。

2.6.6 合理减少测试的工作量

花费更多的精力在测试上，就意味着需要更多的项目费用，没有哪个商业公司愿意做这样的决定。因此，作为测试人员，需要注意怎样合理地减少测试工作量，比如合理缩小测试范围。

一些大型的软件系统，要么是国家投入，要么商业运作涉及重大财产。这类软件系统的质量重于“泰山”，所以对测试的要求非常严格，有时对测试的投入要远远高于对设计和编程的投入，这样的系统毕竟是少数。常见的软件，开发商对测试的投入都是有限度的，如果测试的花费过高，导致开发软件的利润很少甚至赔本，没有哪个软件开发公司愿意做这样的事情，所以，有效地降低软件测试费用是企业普遍关心的问题。降低软件测试费用有两种常用的方法。

1. 减少冗余和无价值测试

这种方法精简了测试工作，且不会影响软件的质量。例如，白盒测试和黑盒测试方法虽然不同，但也有相似之处，在很多地方，白盒测试和黑盒测试会产生相同的效果，这种测试是多余的。一般来说，白盒测试要编写测试驱动程序、逐步跟踪源程序，比黑盒测试烦琐，如果能发现白盒测试的冗余，就能在很大程度上降低成本。在集成测试和系统测试阶段，可能也

需要执行多次的“回归测试”，每次“回归测试”都存在不少的冗余，应该设法找到不必要的重复测试工作。

2. 减少测试阶段

比方说不做单元测试和集成测试，仅仅做一次系统测试，但是这样不能保证软件的质量，看似降低了测试代价，实际上代价并没有降低，而是转移了，后续软件维护的代价会大大增加。

2.6.7　制定测试计划

合理的测试计划有助于测试工作顺利有序地进行，因此要求在对软件进行测试之前所做的测试计划中，应该结合多种针对性强的测试方法，列出所有可使用的资源，建立一个正确的测试目标，本着严谨、准确的原则，周到细致地做好测试前期的准备工作，避免测试的随意性。特别是要科学合理地安排测试时间，留出一定的机动时间，以防意外状况发生时出现测试时间不够用，致使很多测试工作不能正常进行的情况。

在整个软件开发过程中，软件测试工作可能要花费整个项目成本的一半。从项目预算和时间进度安排的角度来看，测试计划和测试工作量估计的有效性对整个测试工作的成败起关键性的作用。图 2-2 是一个测试计划活动图，测试计划活动的产品就是测试计划，可以是一份文档，由测试团队、开发团队和项目管理层进行复查。在制定测试计划过程中，核心活动主要有以下几个方面。

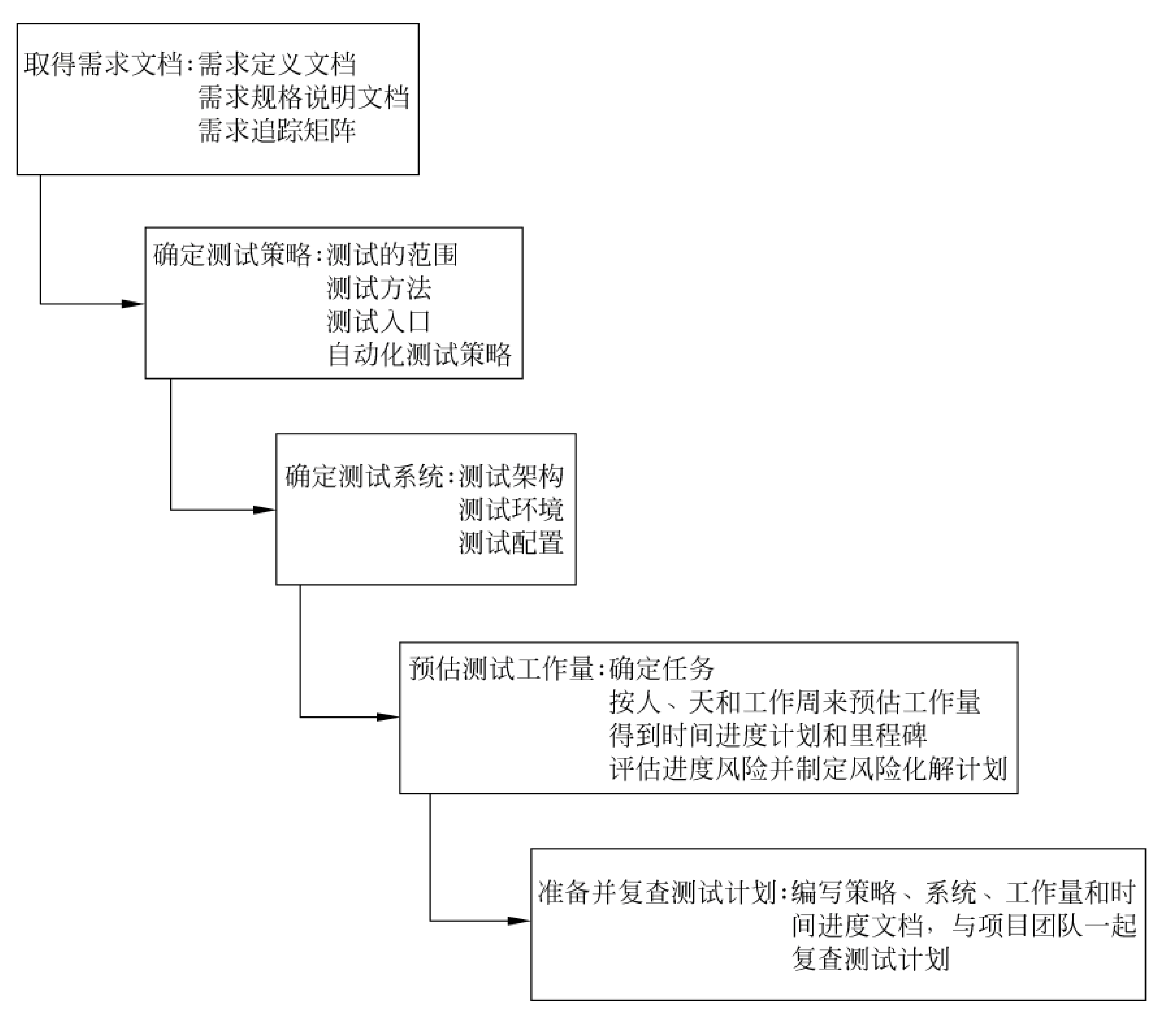


图 2-2　测试计划活动

2.6.8 编写系统测试计划文档

测试计划是描述软件测试努力的目标、范围、方法和焦点的文档。完整的文档将有助于测试组之外的人理解为什么要进行软件正确性检测以及如何进行检测。测试计划应当足够完整，但也不需要太过详细。下面是一些可能包含在测试计划中的内容。

- 标题。
- 确定软件的版本号。
- 修订文档历史(包括作者、日期和批示)。
- 目录表。
- 文档的目的和适合的读者群。
- 测试的目的。
- 软件产品概述。
- 相关文档列表(如需求、设计文档、其他测试计划等)。
- 相关的标准或合法需求。
- 相关的命名规范和标识符规范。
- 整个软件项目组织和人员/联系信息/责任。
- 假设和依赖关系。
- 项目风险信息。
- 测试优先级和焦点。
- 测试范围和限制。
- 测试提纲(就是对测试过程的一个分解，包括测试类型、特点、功能性、过程、系统、模块等)。
- 测试环境设置和配置问题。
- 数据库设置需求。
- 概述系统日志/错误日志/其他性能。
- 有助于测试者跟踪问题根源的具体软硬件工具的论述。
- 使用的测试工具(包括版本、补丁等)。
- 使用的项目测试度量。
- 报告需求和测试可传递性。
- 软件入口和出口准则。
- 初始的理性测试阶段和标准。
- 测试终止和重新开始的标准。
- 人员安排。
- 测试地点。
- 用到的测试外的组织(它们的目的、责任、可传递性、联系人和协作问题)。
- 相关的财产、分类、安全性和许可证问题。
- 附录(包括词汇表、缩略语等)。

测试计划撰写完毕后，应当请有关人员，如项目经理对其进行审批，表 2-2 为测试计划的审核表示例。

在不同公司，对于不同测试计划的审核会有所差别。

表 2-2　测试计划审核表

主要审查项	结　论
测试范围与目标明确吗？	
测试的方法合理吗？	
测试环境具有代表性吗？	
测试工具有效吗？	
测试的开始与结束准则合理吗？	
应递交的文档充分吗？时间合理吗？	
人员组织合理吗？职责明确吗？	
任务分配合理吗？进度安排合理吗？	
审批意见： 签名： 日期：	

2.7　本章小结

本章重点介绍了如何制定测试计划以及如何对测试项目进行管理。在制定测试计划时，需要把握这样几个重点内容：测试策略、定义和建立测试环境、管理测试过程、编写测试文档。

作为初学者，可能并不需要有管理测试工作的能力，但只有先建立这样的意识，才有可能逐步成长为优秀的测试人员。

习题 2

1. 制定软件测试计划的作用有哪些？
2. 制定测试计划的原则是什么？
3. 衡量一份好的测试计划书的标准有哪些？
4. 制定测试计划的主要步骤有哪些？如何确定测试范围？
5. 在制定测试计划时，应该和项目组的哪些成员交流？是否需要了解项目的整体计划？
6. 测试计划文档中的内容有哪些？
7. 只要搭建正确的测试环境，并且拥有足够的资源，就能够保证测试项目的成功，这种观点正确吗？
8. 既然事先制定了项目计划和规则，就要始终严格按照计划和规则办事，测试执行中是否果真如此？

第3章 软件测试的基本技术

软件测试的方法多种多样，可以从不同角度加以分类：从是否需要执行被测软件的角度，分为静态测试和动态测试；从是针对系统的外部功能还是针对系统的内部结构的角度，分为黑盒测试和白盒测试；从软件测试的策略和过程的角度，分为单元测试、集成测试、确认测试、系统测试和验收测试等。本章主要介绍静态测试和动态测试以及黑盒测试和白盒测试。

3.1 软件测试技术的分类

3.1.1 从是否需要执行被测软件的角度分类

从是否需要执行被测软件的角度，软件测试可分为静态测试（static testing）和动态测试（dynamic testing）。

静态测试顾名思义就是通过对被测程序的静态审查，发现代码中潜在的错误。它一般用人工方式脱机完成，故亦称人工测试或代码评审（code review）；也可借助于静态分析器在计算机上以自动方式进行检查，但不要求程序本身在计算机上运行。按照评审的不同组织形式，代码评审又可分为代码会审、走查、办公桌检查以及同行评分 4 种。对某个具体的程序，通常只使用一种评审方式。

动态测试是通常意义上的测试，即通过使用和运行被测软件，发现潜在错误。动态测试的对象必须是能够由计算机真正运行的被测试程序，它包含黑盒测试和白盒测试。

3.1.2 从软件测试用例设计方法的角度分类

从软件测试用例设计方法的角度，可分为黑盒测试（black-box testing）和白盒测试（white-box testing）。

黑盒测试是一种从用户角度出发的测试，又称为功能测试、数据驱动测试或基于规格说明的测试。使用这种方法进行测试时，把被测试程序当作一个黑盒，忽略程序内部的结构特性，测试者在只知道该程序输入和输出之间的关系或程序功能的情况下，依靠能够反映这一关系和程序功能需求规格的说明书，来确定测试用例和推断测试结果的正确性。简单地说，若测试用例的设计是基于产品的功能，目的是检查程序各个功能是否实现，并检查其中的功能错误，则这种测试方法称为黑盒。

白盒测试则基于产品的内部结构来进行测试,检查内部操作是否按规定执行,软件各个部分功能是否得到充分利用。白盒测试又称为结构测试、逻辑驱动测试或基于程序的测试,即根据被测程序的内部结构设计测试用例,测试者需要预先了解被测试程序的结构。

3.1.3　从软件测试的策略和过程的角度分类

按照软件测试的策略和过程分类,软件测试可分为单元测试(unit testing)、集成测试(integration testing),确认测试(validation testing),系统测试(system testing)和验收测试(verification testing)。

单元测试是针对每个单元的测试,是软件测试的最小单位,它旨在确保每个模块能正常工作。单元测试主要采用白盒测试方法,以发现内部错误。

集成测试是对已测试过的模块进行组装,进行集成测试的目的主要在于检验与软件设计相关的程序结构问题。在集成测试过程中,测试人员采用黑盒测试和白盒测试两种方法,以验证多个单元模块集成到一起后是否能够协调工作。

确认测试是检验所开发的软件能否满足所有功能和性能需求的最后手段,通常采用黑盒测试方法。

系统测试的主要任务是检测被测软件与系统的其他部分的协调性,通常采用黑盒测试方法。

验收测试是软件产品质量的最后一关,在这一环节,测试主要从用户的角度着手,参与者主要是用户以及少量的程序开发人员,通常采用黑盒测试方法。

3.2　静态测试和动态测试

3.2.1　静态测试

如前所述,静态测试和动态测试的一个重要区别就是是否需要运行被测程序。图 3-1 是静态测试与动态测试的比喻图。

静态测试方法的主要特征是在测试源程序时,计算机并不真正运行被测试的程序,只对被测程序进行特性分析。因此,静态方法常称为“分析”,静态分析是对被测程序进行特性分析的一些方法的总称。所谓静态分析,就是不需要执行所测试的程序,而只是通过扫描程序正文,对程序的数据流和控制流等信息进行分析,找出系统的缺陷,得出测试报告。

踩油门
看车漆
打开前盖检查 } 静态测试

发动汽车
听听发动机声音
上路行驶 } 动态测试

图 3-1　静态测试与动态测试比喻图

静态测试包括代码检查、静态结构分析、代码质量度量等。它可以由人工进行,充分发挥人的逻辑思维优势,也可以借助软件工具自动进行。

通常在静态测试阶段进行以下一些测试活动:

(1) 检查算法的逻辑正确性,确定算法是否实现了所要求的功能;

(2) 检查模块接口的正确性,确定形参的个数、数据类型、顺序是否正确,确定返回值类

型及返回值的正确性；

(3) 检查输入参数是否有合法性检查。如果没有合法性检查，则应确定该参数是否不需要合法性检查，否则应加上参数的合法性检查；

(4) 检查调用其他模块的接口是否正确，检查实参类型、实参个数是否正确，返回值是否正确。若被调用模块出现异常或错误，程序是否有适当的出错处理代码；

(5) 检查是否设置了适当的出错处理，以便在程序出错时，能对出错部分重做安排，保证其逻辑的正确性；

(6) 检查表达式、语句是否正确，是否含有二义性。例如，检查表达式或运算符的优先级：＜＝、＝、＞＝、＆＆、||、＋＋、－－等；

(7) 检查常量或全局变量使用是否正确；

(8) 检查标识符的使用是否规范、一致，变量命名是否能够做到望名知义、简洁、规范和易记；

(9) 检查程序风格的一致性、规范性，代码是否符合行业规范，是否所有模块的代码风格一致、规范；

(10) 检查代码是否可以优化，算法效率是否最高；

(11) 检查代码注释是否完整，是否正确反映了代码的功能，并查找错误的注释。

静态分析的差错分析功能是编译程序所不能替代的，编译系统虽然能发现某些程序错误，但这些错误远非软件中存在的大部分错误。目前，已经开发了一些静态分析系统作为软件静态测试的工具，静态分析已被当作一种自动化的代码校验方法。

3.2.2 动态测试

动态测试方法是通过源程序运行时所体现出来的特征，来进行执行跟踪、时间分析以及测试覆盖等方面的测试。动态测试借由真正运行被测程序，在执行过程中，输入有效的测试用例，对输入与输出进行分析，以达到检测的目的。

动态测试方法的基本步骤如下：

(1) 选取定义域的有效值，或选取定义域外的无效值。

(2) 对已选取值决定预期的结果。

(3) 用选取值执行程序。

(4) 将执行结果与预期的结果相比，不吻合则说明程序有错。

不同的测试方法各自的目标和侧重点不一样，在实际工作中要将静态测试和动态测试结合起来，以达到更加完美的效果。

3.3 黑盒测试方法

3.3.1 黑盒测试方法概述

黑盒测试(Black-box Testing)又称为功能测试、数据驱动测试和基于规格说明的测试。是一种从用户观点出发的测试，主要以软件规格说明书为依据，是对程序功能和程序接口进

行的测试。

黑盒测试的基本观点是：任何程序都可以看作是从输入定义域映射到输出值域的函数过程。黑盒测试将被测程序视为一个打不开的黑盒子，黑盒中的内容（实现过程）完全不知道，只明确盒子要做到什么。黑盒测试作为软件功能的测试手段，是重要的测试方法。它并不涉及程序内部结构和内部特性，主要根据规格说明，只依靠被测程序输入和输出之间的关系或程序的功能来设计测试用例。

黑盒测试是以用户的观点，从输入数据与输出数据的对应关系出发进行测试的，它不涉及程序的内部结构。很明显，如果外部特性本身有问题或规格说明书的规定有误，用黑盒测试方法是发现不了的。黑盒测试方法着重测试软件的功能需求，是在程序接口上进行测试，主要是为了发现以下错误：

（1）是否有不正确的功能或是遗漏的功能；

（2）接口能否正确地接收输入数据并产生正确的输出结果；

（3）是否有数据结构错误或外部信息访问错误；

（4）性能是否能够满足要求；

（5）是否有程序初始化和终止方面的错误。

黑盒测试有两个显著的特点：

（1）黑盒测试不考虑软件的具体实现过程，当在软件实现的过程发生变化时，测试用例仍然可以使用；

（2）黑盒测试用例的设计可以和软件实现同时进行，这样能够压缩总的开发时间。

黑盒测试不仅能够找到大多数其他测试方法无法发现的错误，而且一些外购软件、参数化软件包以及某些自动生成的软件，由于无法得到源程序，在一些情况下只能选择黑盒测试。

黑盒测试有两种基本方法，即通过测试和失败测试。

在进行通过测试时，实际上是确认软件能做什么，而不会去考验其能力如何，软件测试人员只运用最简单、最直观的测试案例。在设计和执行测试案例时，总是先要进行通过测试，验证软件的基本功能是否都已实现。

在确信了软件正确运行之后，就可以采取各种手段搞垮软件来找出缺陷，这种纯粹为了破坏软件而设计和执行的测试案例，被称为失败测试或迫使出错测试。

黑盒测试的具体技术方法主要包括边界值分析法、等价类划分法、因果图法、决策表法等。这些方法都比较实用，在设计具体的测试方案时要针对开发项目的特点进行适当的选择。

3.3.2　等价类划分法

1. 等价类划分法概述

等价类划分法是黑盒测试用例设计中一种常用的设计方法，它将不能穷举的测试过程进行合理分类，从而保证设计出来的测试用例具有完整性和代表性。

等价类划分法把所有可能的输入数据，即程序的输入域划分成若干部分（子集），然后从每一个子集中选取少数具有代表性的数据作为测试用例。所谓等价类是指输入域的某个子

集合,所有等价类的并集就是整个输入域,在等价类中,各个输入数据对于揭露程序中的错误都是等效的,它们具有等价特性。因此,测试某个等价类的代表值就等价于对这一类中其他值的测试。也就是说,如果某一类中的一个例子发现了错误,这一等价类中的其他例子也能发现同样的错误;反之,如果某一类中的一个例子没有发现错误,则这一类中的其他例子也不会查出错误。

软件不能只接收合理有效的数据,也要具有处理异常数据的功能,这样的测试才能确保软件具有更高的可靠性。因此,在划分等价类的过程中,不但要考虑有效等价类划分,同时也要考虑无效等价类划分。

有效等价类是指对软件规格说明来说,合理、有意义的输入数据所构成的集合,利用有效等价类可以检验程序是否满足规格说明所规定的功能和性能。无效等价类则和有效等价类相反,即不满足程序输入要求或者无效的输入数据所构成的集合。利用无效等价类可以检验程序异常情况的处理。

使用等价类划分法设计测试用例,首先必须在分析需求规格说明的基础上划分等价类,然后列出等价类表。

在确立了等价类之后,可建立等价类表,列出所有划分出的等价类,如表 3-1 所示。

表 3-1 等价类表

输 入 条 件	有效等价类	无效等价类
…	…	…
…	…	…

再根据已列出的等价类表,按以下步骤确定测试用例:

(1) 为每一个等价类规定一个唯一的编号;

(2) 设计一个新的测试用例,使其尽可能多地覆盖有效等价类,重复这个过程,直至所有的有效等价类均被测试用例所覆盖;

(3) 设计一个新的测试用例,使其仅覆盖一个无效等价类,重复这个过程,直至所有的无效等价类均被测试用例所覆盖。

以三角形问题为例,输入条件是:三个数,分别作为三角形的三条边;都是整数;取值范围在 1～100 之间。认真分析上述的输入条件,可以得出相关的等价类表(包括有效等价类和无效等价类),如表 3-2 所示。

表 3-2 三角形问题等价类表

<table>
<tr><th>输 入 条 件</th><th>等价类编号</th><th>有效等价类</th><th>等价类编号</th><th>无效等价类</th></tr>
<tr><td rowspan="3">三个数</td><td rowspan="3">1</td><td rowspan="3">三个数</td><td>4</td><td>只有一条边</td></tr>
<tr><td>5</td><td>只有两条边</td></tr>
<tr><td>6</td><td>多于三条边</td></tr>
<tr><td rowspan="3">整数</td><td rowspan="3">2</td><td rowspan="3">整数</td><td>7</td><td>一边为非整数</td></tr>
<tr><td>8</td><td>两边为非整数</td></tr>
<tr><td>9</td><td>三边为非整数</td></tr>
</table>

续表

输入条件	等价类编号	有效等价类	等价类编号	无效等价类
取值范围在1～100之间	3	$1\leqslant a\leqslant 100$ $1\leqslant b\leqslant 100$ $1\leqslant c\leqslant 100$	10	一边为0
			11	两边为0
			12	三边为0
			13	一边小于0
			14	两边小于0
			15	三边小于0
			16	一边大于100
			17	两边大于100
			18	三边大于100

2. 常见等价类划分形式

针对是否对无效数据进行测试，可以将等价类测试分为标准等价类测试、健壮等价类测试以及对等区间划分。

1）标准等价类测试

标准等价类测试不考虑无效数据值，测试用例使用每个等价类中的一个值。通常，标准等价类测试用例的数量和最大等价类中元素的数目相等。

以三角形问题为例，要求输入三个整数 a、b、c，分别作为三角形的三条边，其取值范围在1～100之间，判断由这三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形（包括直角三角形）或是非三角形。在多数情况下，是从输入域划分等价类，但对于三角形问题，从输出域来定义等价类是最简单的划分方法。

因此，利用这些信息可以确定下列值域等价类：

$R_1=\{\langle a,b,c\rangle$：边为 a、b、c 的等边三角形$\}$

$R_2=\{\langle a,b,c\rangle$：边为 a、b、c 的等腰三角形$\}$

$R_3=\{\langle a,b,c\rangle$：边为 a、b、c 的一般三角形$\}$

$R_4=\{\langle a,b,c\rangle$：边 a、b、c 不构成三角形$\}$

4个标准等价类测试用例如表3-3所示。

表3-3　三角形问题的标准等价类测试用例

测试用例	a	b	c	预期输出
Test Case 1	10	10	10	等边三角形
Test Case 2	10	10	5	等腰三角形
Test Case 3	3	4	5	一般三角形
Test Case 4	1	1	5	不构成三角形

2）健壮等价类测试

健壮等价类测试主要的出发点是考虑无效等价类，对有效输入，测试用例从每个有效等价类中取一个值；对无效输入，一个测试用例有一个无效值，其他值均取有效值。

3）对等区间划分

对等区间划分是测试用例设计的非常规形式化的方法，它将被测对象的输入/输出划分成一些区间，被测软件对一个特定区间的任何值都是等价的。形成测试区间的数据不只限于函数/过程的参数，也可以是程序可以访问的全局变量、系统资源等，这些变量或资源可以是以时间形式存在的数据，或以状态形式存在的输入/输出序列。

对等区间划分假定位于单个区间的所有值对测试都是对等的，应为每个区间的一个值设计一个测试用例，举例说明如下：

平方根函数要求输入值为 0 或大于 0，而后才返回输入数的平方根；当输入值小于 0 时，显示错误信息“平方根错误，输入值小于 0”，并返回 0。考虑平方根函数的测试用例区间，可以划分出两个输入区间和两个输出区间，如表 3-4 所示。

表 3-4　区间划分

输入区间		输出区间	
ⅰ	＜0	A	＞＝0
ⅱ	＞＝0	B	Error

通过分析，可以用两个测试用例来测试上述 4 个区间：

```
测试用例 1: 输入 4,返回 2                                  //区间 ⅱ 和 A
测试用例 2: 输入 - 4,返回 0,输出"平方根错误,输入值小于 0"    //区间 ⅰ 和 B
```

上例的对等区间划分是非常简单的，当软件变得更加复杂时，对等区间的确定也会越难，区间之间的相互依赖性就越强，使用对等区间划分设计测试用例技术的难度会增加。

3.3.3　边界值分析法

1. 边界值分析法

边界值分析法（Boundary Value Analysis，BVA）是一种补充等价类划分法的测试用例设计技术，不同于等价类划分法选择等价类的任意元素，它选择等价类的边界来设计测试用例。在测试过程中，测试人员可能会忽略边界值的条件，而软件设计中大量的错误往往就发生在输入或输出范围的边界上，而非输入输出范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。在实际的软件设计过程中，会涉及到大量的边界值条件和过程，这里有一个简单的 VB 程序例子：

```
Dim data(10) as Integer
Dim i as Integer
For i = 1 to 10
    data(i) = 1
Next i
```

这个程序的目标是创建一个拥有 10 个元素的一维数组，看似合理，但是，在大多数 Basic 语言中，当一个数组被定义时，其第一个元素所对应的数组下标是 0 而不是 1。由此，上述程序运行结束后，数组中成员的实际赋值情况如下：

```
data(0) = 0,data(1) = 1,data(2) = 1, … ,data(10) = 1
```

这时,如果其他程序员未注意而使用了这个数组,就可能会造成软件的缺陷或者产生错误。

使用边界值分析方法设计测试用例,首先应确定边界情况。通常输入和输出等价类的边界,就是应着重测试的边界情况,应当选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据,而不是选取等价类中的典型值或任意值作为测试数据。

在应用边界值分析法设计测试用例时,应遵循以下几条原则:

(1) 如果输入条件规定了值的范围,则应该选取刚达到这个范围的边界值,以及刚刚超过这个范围边界的值作为测试输入数据。

(2) 如果输入条件规定了值的个数,则用最大个数、最小个数、比最小个数少1、比最大个数多1的数作为测试数据。

根据规格说明的每一个输出条件,分别使用以上两个原则。

(3) 如果程序的规格说明给出的输入域或者输出域是有序集合(如有序表、顺序文件等),则应选取集合的第一个元素和最后一个元素作为测试用例。

(4) 如果程序中使用了一个内部数据结构,则应当选择这个内部数据结构的边界值作为测试用例。

2. 边界条件与次边界条件

由于边界值分析法是对输入的边界值进行测试,在测试用例设计中,需要对输入的条件进行分析并且找出其中的边界值条件,通过对这些边界值的测试来查出更多的错误。提出边界条件时,一定要测试临近边界的有效数据,测试最后一个可能有效的数据,同时测试刚超过边界的无效数据。通常情况下,软件测试所包含的边界检验有几种类型:数值、字符、位置、数量、速度、尺寸等。在设计测试用例时要考虑边界检验的类型特征:第一个/最后一个、开始/完成、空/满、最大值/最小值、最快/最慢、最高/最低、最长/最短等,这些不是确定的列表,而是一些可能出现的边界条件。

在多数情况下,边界值条件是基于应用程序的功能设计需要考虑的因素,可以从软件的规格说明或常识中得到,也是最终用户通常最容易发现问题的部分。然而,在测试用例设计过程中,某些边界值条件是不需要呈现给用户的,或者说用户很难注意到这些问题,但这些边界条件同时确实属于检验范畴内的边界条件,称为内部边界值条件或次边界值条件。主要有下面几种。

1) 数值的边界值检验

计算机是基于二进制进行工作的,因此,任何数值运算都有一定的范围限制,如表3-5所示。

表3-5　计算机数值运算的范围

项	范围或值
位(bit)	0或1
字节(byte)	0～255
字(word)	0～65,535(单字)或0～4,294,967,295(双字)

续表

项	范 围 或 值
千(K)	1024
兆(M)	1 048 576
吉(G)	1 073 741 824
太(T)	1 099 511 627 776

例如对字节进行检验,边界值条件可以设置成 254、255 和 256。

2) 字符的边界值检验

在字符的编码方式中,ASCII 和 Unicode 是比较常见的编码方式,表 3-6 中列出了一些简单的 ASCII 码对应表。

表 3-6 字符的 ASCII 码对应表

字　　符	ASCII 码值	字　　符	ASCII 码值
空(null)	0	A	65
空格(space)	32	a	97
斜杠(/)	47	左中括号([)	91
0	48	Z	122
冒号(:)	58	Z	90
@	64	单引号(')	96

3. 边界值分析法测试用例

还是以三角形问题为例,要求输入三个整数 a、b、c 分别作为三角形的三条边,取值范围在 1～100 之间,判断由三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形(包括直角三角形)或是三角形。表 3-7 给出了边界值分析测试用例。

表 3-7 边界值分析测试用例

测 试 用 例	a	b	c	预 期 输 出
Test Case 1	1	50	50	等腰三角形
Test Case 2	2	50	50	等腰三角形
Test Case 3	50	50	50	等边三角形
Test Case 4	99	50	50	等腰三角形
Test Case 5	100	50	50	非三角形
Test Case 6	50	1	50	等腰三角形
Test Case 7	50	2	50	等腰三角形
Test Case 8	50	99	50	等腰三角形
Test Case 9	50	100	50	非三角形
Test Case 10	50	50	1	等腰三角形
Test Case 11	50	50	2	等腰三角形
Test Case 12	50	50	99	等腰三角形
Test Case 13	50	50	100	非三角形

3.3.4　决策表法

1. 决策表法简介

在所有的黑盒测试方法中，基于决策表（也称判定表）的测试是最为严格、最具有逻辑性的测试方法。决策表是分析和表达多个逻辑条件下执行不同操作的有力工具，由于决策表可以把复杂的逻辑关系和多种条件组合的情况表达得既具体又明确，在程序设计发展的初期，决策表就已被当作编写程序的辅助工具了。决策表通常由4个部分组成，如图3-2所示。

条件桩：列出了问题的所有条件，通常认为列出条件的先后次序无关紧要。

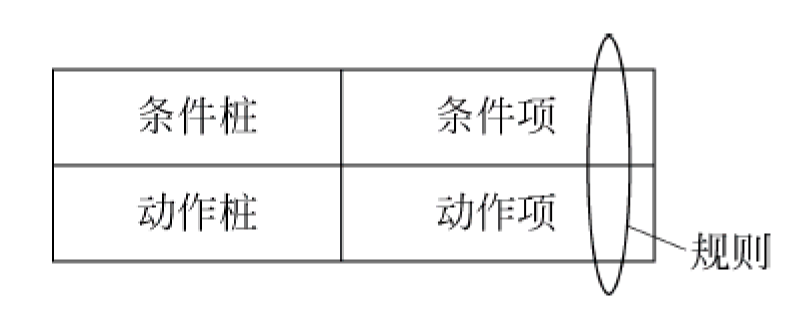


图3-2　决策表的组成

- 动作桩：列出了问题规定的可能采取的操作，这些操作的排列顺序没有约束。
- 条件项：针对条件桩给出的条件列出所有可能的取值。
- 动作项：与条件项紧密相关，列出在条件项的各组取值情况下应该采取的动作。

任何一个条件组合的特定取值及其相应要执行的操作称为一条规则，在决策表中贯穿条件项和动作项的一列就是一条规则。显然，决策表中列出多少组条件取值，也就有多少条规则，即条件项和动作项有多少列。根据软件规格说明，建立决策表的步骤如下：

(1) 确定规则的个数。假如有 n 个条件，每个条件有两个取值，故有 2^n 种规则。

(2) 列出所有的条件桩和动作桩。

(3) 填入条件项。

(4) 填入动作项，得到初始决策表。

(5) 化简，合并相似规则（相同动作）。

以下列问题为例给出构造决策表的具体过程：

如果某产品销售好并且库存低，则增加该产品的生产；如果该产品销售好，但库存量不低，则只继续生产；若该产品销售不好，但库存量低，也继续生产；若该产品销售不好，且库存量不低，则停止生产。

解法如下：

首先确定规则的个数。对于本题有2个条件（销售、库存），每个条件可以有两个取值，故有2×2=4种规则。然后列出所有的条件桩和动作桩，填入条件项和动作项，即得到初始决策表，如表3-8所示。

每种测试方法都有适用的范围，决策表法适用于下列情况：

- 规格说明以决策表形式给出，或很容易转换成决策表；
- 条件的排列顺序不会也不应影响执行哪些操作；
- 规则的排列顺序不会也不应影响执行哪些操作；
- 每当某一规则的条件已经满足，并确定要执行的操作后，不必检验其他规则；
- 如果某一规则得到满足要执行多个操作，这些操作的执行顺序无关紧要。

表 3-8 产品销售问题的决策表

选项 \ 规则	1	2	3	4
条件：				
C_1：销售好？	T	T	F	F
C_2：库存低？	T	F	T	F
动作：				
a_1：增加生产	√			
a_2：继续生产		√	√	
a_3：停止生产				√

2. 决策表法的应用

决策表法最突出的优点是，能够将复杂的问题按照各种可能的情况全部列举出来，简明并避免遗漏，因此，利用决策表能够设计出完整的测试用例集合。运用决策表设计测试用例时，可以将条件理解为输入，将动作理解为输出。

以三角形问题为例，要求输入三个整数 a、b、c 分别作为三角形的三条边，取值范围在 1～100 之间，判断由三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形（包括直角三角形）或是非三角形。

分析如下：

(1) 确定规则的个数。例如，三角形问题的决策表有 4 个条件，每个条件可以取两个值（真值和假值），所以应该有 $2^4=16$ 种规则。

(2) 列出所有条件桩和动作桩。

(3) 填写条件项。

(4) 填写动作项，从而得到初始决策表，如表 3-9 所示。

(5) 简化决策表，合并相似规则后得到三角形问题的简化决策表，如表 3-10 所示。

根据决策表 3-10，即可设计测试用例，如表 3-11 所示。

表 3-9 三角形问题的初始决策表

选项 \ 规则	1	2	3	4	5	6	7	8
条件：								
C_1：a、b、c 构成一个三角形？	F	F	F	F	F	F	F	F
C_2：$a=b$?	T	T	T	T	F	F	F	F
C_3：$b=c$?	T	T	F	F	T	T	F	F
C_4：$a=c$?	T	F	T	F	T	F	T	F
动作：								
a_1：非三角形	√	√	√	√	√	√	√	√
a_2：一般三角形								
a_3：等腰三角形								
a_4：等边三角形								
a_5：不可能								

续表

规则 选项	9	10	11	12	13	14	15	16
条件：								
C_1：a、b、c 构成一个三角形？	T	T	T	T	T	T	T	T
C_2：$a=b$?	T	T	T	T	F	F	F	F
C_3：$b=c$?	T	T	F	F	T	T	F	F
C_4：$a=c$?	T	F	T	F	T	F	T	F
动作：								
a_1：非三角形								
a_2：一般三角形								√
a_3：等腰三角形				√		√	√	
a_4：等边三角形	√							
a_5：不可能		√	√		√			

表 3-10　三角形问题的简化决策表

规则 选项	1～8	9	10	11	12	13	14	15	16
条件：									
C_1：a、b、c 构成一个三角形？	F	T	T	T	T	T	T	T	T
C_2：$a=b$?	—	T	T	T	T	F	F	F	F
C_3：$b=c$?	—	T	T	F	F	T	T	F	F
C_4：$a=c$?	—	T	F	T	F	T	F	T	F
动作：									
a_1：非三角形	√								
a_2：一般三角形									√
a_3：等腰三角形					√		√	√	
a_4：等边三角形		√							
a_5：不可能			√	√		√			

表 3-11　三角形问题的决策表测试用例

测 试 用 例	a	b	c	预 期 输 出
Test Case 1	10	4	4	非三角形
Test Case 2	4	4	4	等边三角形
Test Case 3	?	?	?	不可能
Test Case 4	?	?	?	不可能
Test Case 5	4	4	5	等腰三角形
Test Case 6	?	?	?	不可能
Test Case 7	5	4	4	等腰三角形
Test Case 8	4	5	4	等腰三角形
Test Case 9	3	4	5	一般三角形

3.3.5 因果图法概述

等价类划分法和边界值分析法都着重考虑输入条件本身，而没有考虑到输入条件的各种组合情况，也没有考虑到各个输入条件之间的相互制约关系。因此，必须考虑采用一种适合多种条件的组合，相应能产生多个动作的形式来进行测试用例的设计，这就需要采用因果图法。因果图法就是一种利用图解法分析输入的各种组合情况，从而设计测试用例的方法，它适合检查程序输入条件的各种组合情况。

在因果图中使用 4 种符号分别表示 4 种因果关系，如图 3-3 所示。用直线连接左右节点，其中左节点 C_i 表示输入状态(或称原因)，右节点 e_i 表示输出状态(或称结果)。C_i 和 e_i 都可取值 0 或 1，0 表示某状态不出现，1 表示某状态出现。

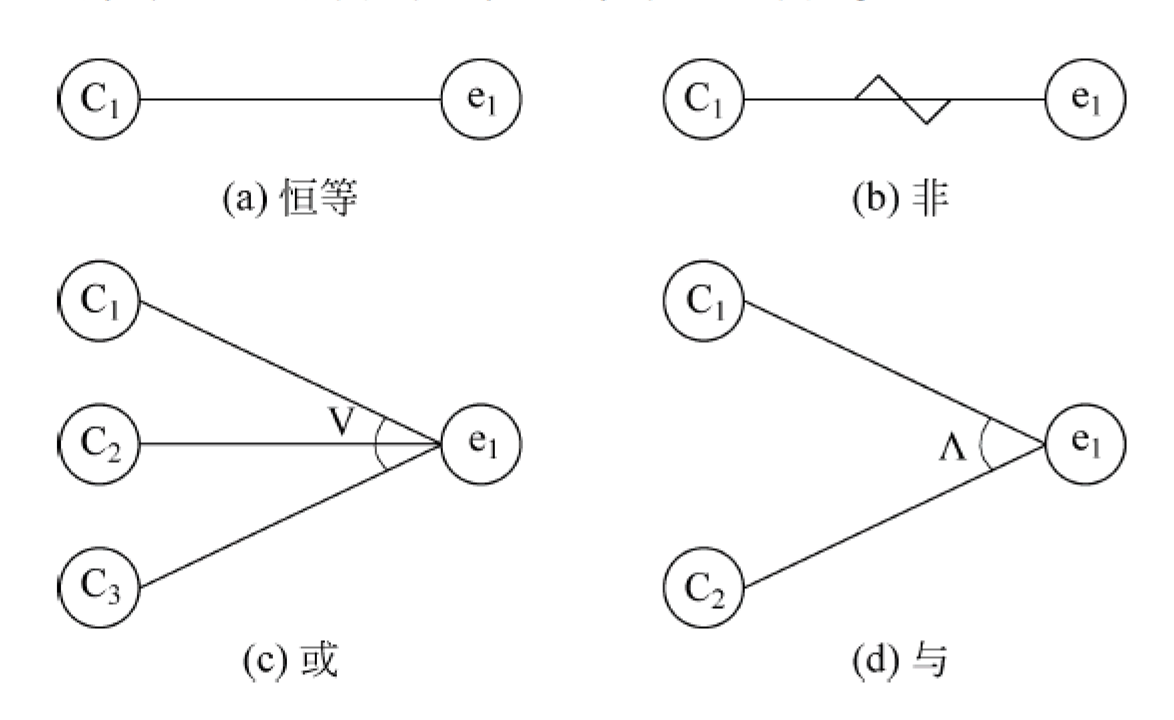


图 3-3 因果图中 4 种因果关系

图 3-3 中各符号的含义如下。

图(a)：表示恒等。若 C_1 是 1，则 e_1 也是 1；若 C_1 是 0，则 e_1 为 0。

图(b)：表示非。若 C_1 是 1，则 e_1 是 0；若 C_1 是 0，则 e_1 为 1。

图(c)：表示或。若 C_1 或 C_2 或 C_3 是 1，则 e_1 是 1；若 C_1、C_2、C_3 全为 0，则 e_1 为 0。

图(d)：表示与。若 C_1 和 C_2 都是 1，则 e_1 是 1；只要 C_1、C_2、C_3 中有一个为 0，则 e_1 为 0。

在实际问题中，输入状态相互之间还可能存在某些依赖关系，我们称之为约束。例如，某些输入条件不可能同时出现，输出状态之间也往往存在约束，在因果图中，以特定的符号标明这些约束，如图 3-4 所示。

图 3-4 中对输入条件的约束如下。

图(a)：表示 E 约束(异)。a 和 b 中最多有一个可能为 1，即 a 和 b 不能同时为 1。

图(b)：表示 I 约束(或)。a、b 和 c 中至少有一个必须是 1，即 a、b 和 c 不能同时为 0。

图(c)：表示 O 约束(唯一)。a 和 b 中必须有一个且仅有一个为 1。

图(d)：表示 R 约束(要求)。a 是 1 时，b 必须是 1，即 a 是 1 时，b 不能是 0。

图(e)：表示 M 约束(强制)：若结果 a 是 1，则结果 b 强制为 0。对输出条件的约束只有 M 约束。

因果图法最终要生成决策表，然后设计测试用例，需要以下几个步骤：

(1) 分析软件规格说明书中的输入输出条件，并且分析出等价类。分析规格说明中的

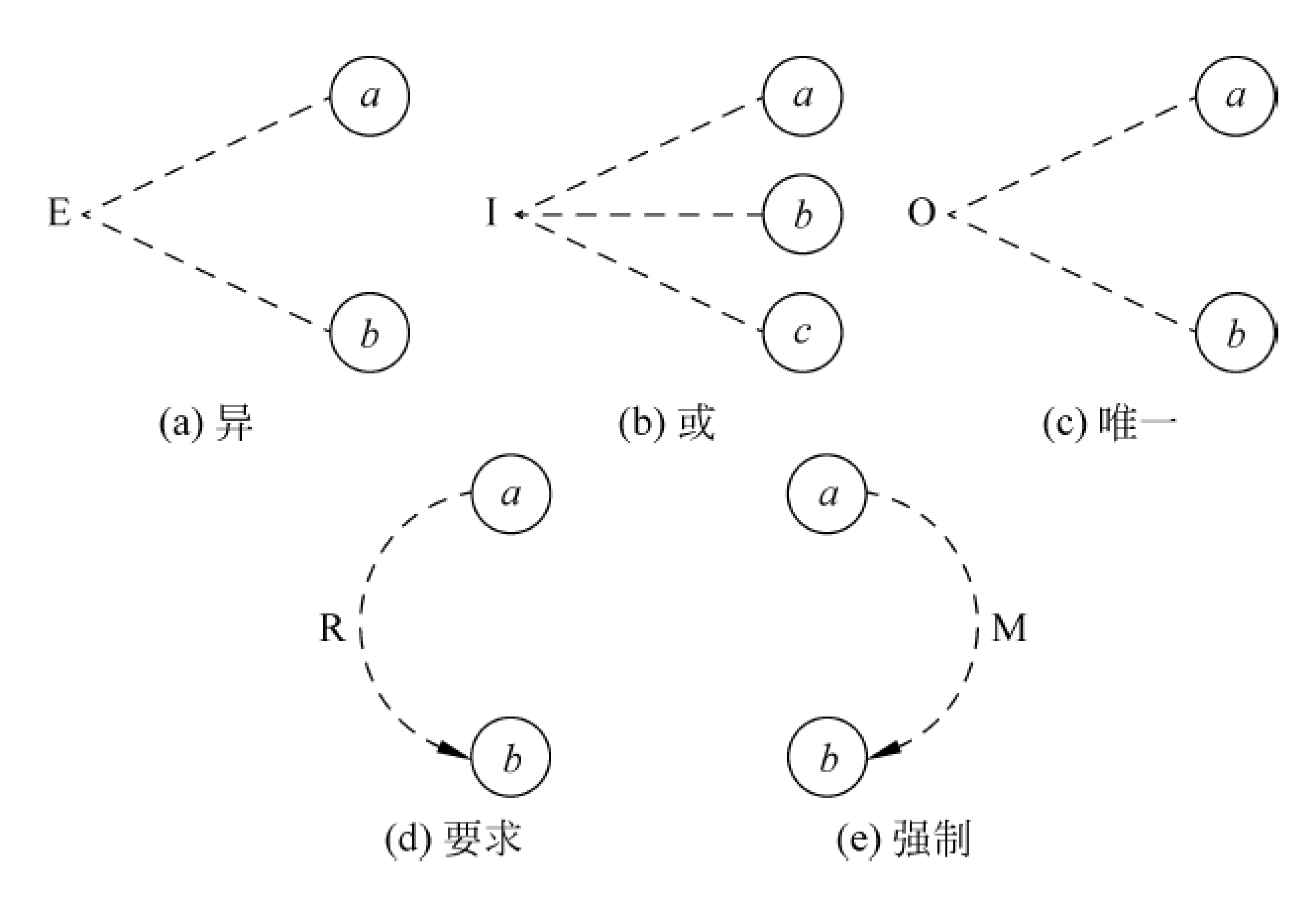


图 3-4　约束符号

语义内容，通过这些语义来找出相对应的输入与输入之间、输入与输出之间的对应关系；

（2）将对应的输入与输入之间、输入与输出之间的关系连接起来，并且将其中不可能的组合情况标注成约束或者限制条件，形成因果图；

（3）将因果图转换成决策表；

（4）将决策表的每一列作为依据，设计测试用例。

上述步骤用图 3-5 表示。

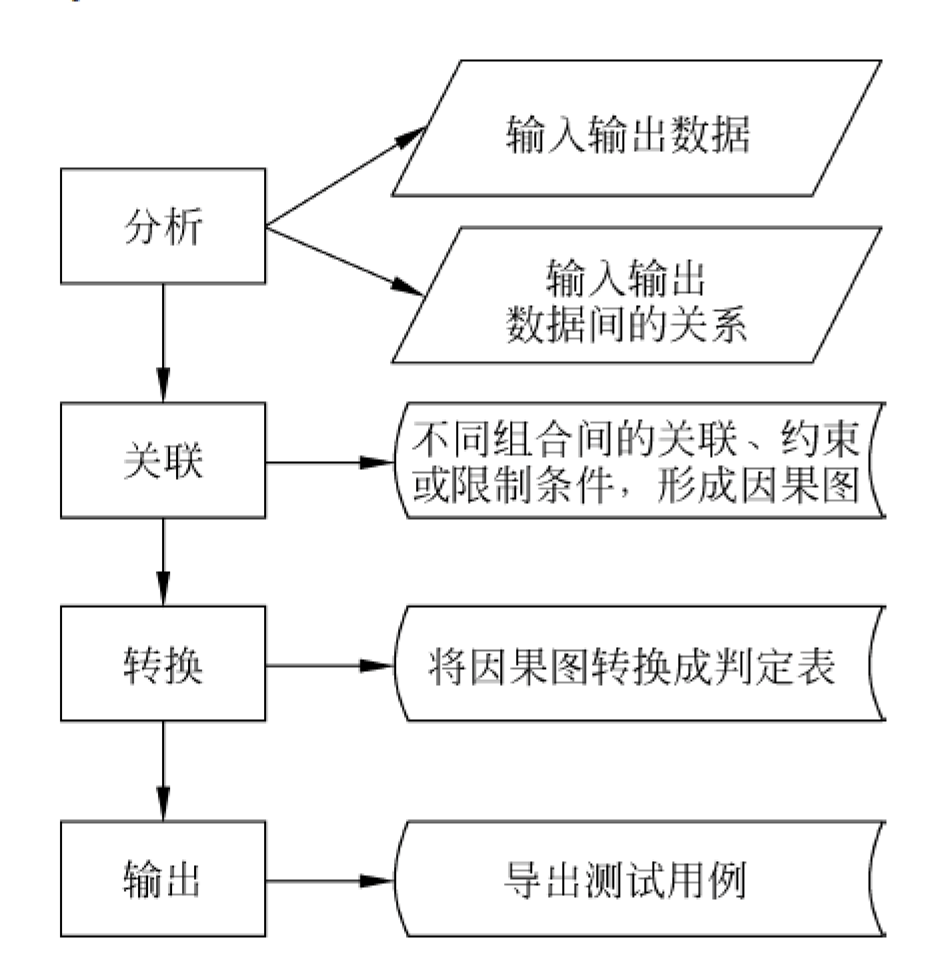


图 3-5　因果图法生成测试用例步骤

因果图生成的测试用例中包括了所有输入数据取真值和假值的情况，而构成的测试用例数目达到最少，其测试用例数目随输入数据数目的增加而线性地增加。

例如某软件规格说明中包含这样的要求：输入的第一个字符必须是 A 或 B，第二个字符必须是一个数字，在此情况下可进行文件的修改；如果第一个字符不正确，则给出信息 L；如果第二个字符不是数字，则给出信息 M。

解法如下：

分析程序的规格说明，列出原因和结果。得

原因：C_1——第一个字符是 A

C_2——第一个字符是 B

C_3——第二个字符是一个数字

结果：e_1——给出信息 L

e_2——修改文件

e_3——给出信息 M

将原因和结果之间的因果关系用逻辑符号连接起来，得到因果图，如图 3-6 所示。编号为 11 的中间节点是导出结果的进一步原因。

因为 C_1 和 C_2 不可能同时为 1，即第一个字符不可能既是 A 又是 B，在因果图上可对其施加 E 约束，得到具有约束的因果图，如图 3-7 所示。

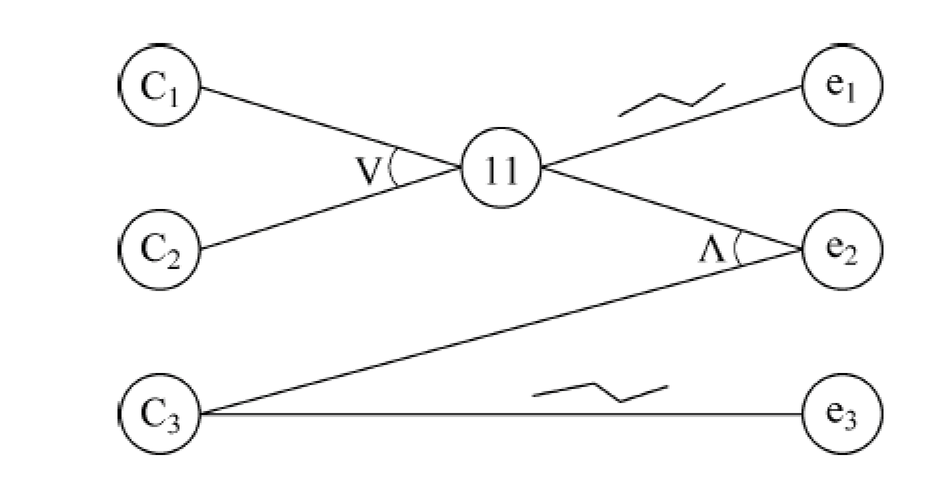


图 3-6　因果图示例

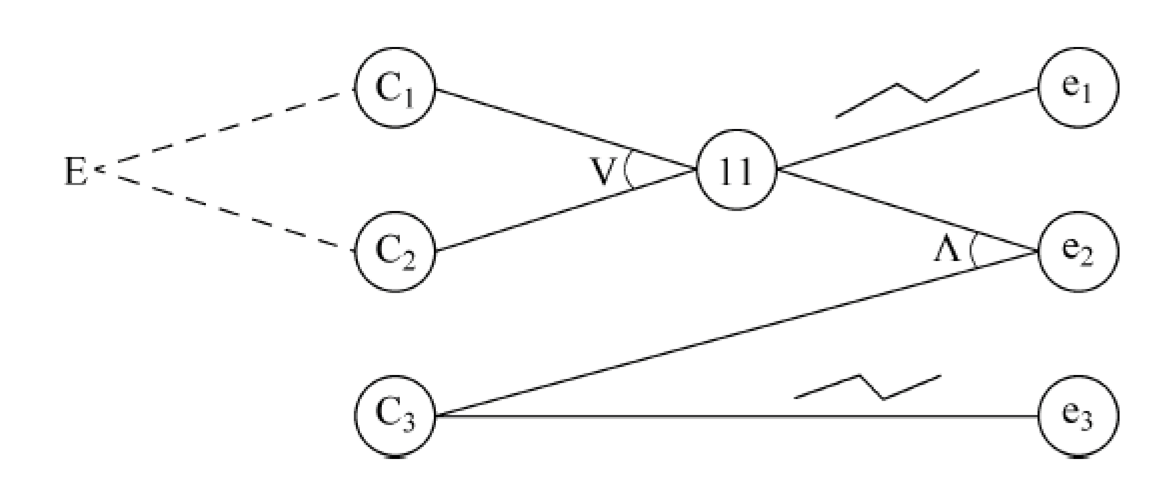


图 3-7　具有约束的因果图

将因果图转换成决策表，得到表 3-12。

表 3-12　决策表

选项 \ 规则		1	2	3	4	5	6	7	8
条件	C_1	1	1	1	1	0	0	0	0
	C_2	1	1	0	0	1	1	0	0
	C_3	1	0	1	0	1	0	1	0
	11			1	1	1	1	0	0
动作	e_1			0	0	0	0	1	1
	e_2			1	0	1	0	0	0
	e_3			0	1	0	1	0	1
	不可能	1	1						
测试用例				A5	A#	B9	B?	X2	Y%

对表 3-12 中的前两种情况,因为 C_1 和 C_2 不可能同时为 1,所以应排除这两种情况,根据此表,可以设计出 6 个测试用例,如表 3-13 所示。

表 3-13 测试用例

编 号	输入数据	预期输出
Test Case 1	A5	修改文件
Test Case 2	A#	给出信息 M
Test Case 3	B9	修改文件
Test Case 4	B?	给出信息 M
Test Case 5	X2	给出信息 L
Test Case 6	Y%	给出信息 L 和信息 M

3.3.6 黑盒测试方法的选择

1. 黑盒测试方法的优缺点

黑盒测试的优点:适用于各个测试阶段;从产品功能角度进行测试;容易入手生成测试数据。

黑盒测试的缺点:某些代码得不到测试;如果规则说明有误,无法发现;不易充分进行测试。

2. 各种黑盒测试方法的选择

为了最大限度地减少测试遗留的缺陷,同时也为了最大限度地发现存在的缺陷,在测试实施之前,测试工程师必须确定将要采用的黑盒测试策略和方法,并以此为依据制定详细的测试方案。通常,一个好的测试策略和测试方法必将给整个测试工作带来事半功倍的效果。如何才能确定好的黑盒测试策略和测试方法呢?通常,在确定黑盒测试方法时,应该遵循以下原则。

(1) 根据程序的重要性和一旦发生故障将造成的损失程度来确定测试等级和测试重点。

(2) 认真选择测试策略,以便能尽可能少地使用测试用例,而发现尽可能多的程序错误。一次完整的软件测试过后,如果程序中遗留的错误仍过多并且严重,则表明该次测试是不足的,而测试不足则意味着让用户承担隐藏错误带来的危险,但测试过度又会带来资源的浪费,因此,测试需要找到一个平衡点。

(3) 进行等价类划分,包括输入条件和输出条件的等价划分,将无限测试变成有限测试,这是减少工作量和提高测试效率的最有效方法。

(4) 在任何情况下都必须使用边界值分析方法,经验表明用这种方法设计出测试用例发现程序错误的能力最强。

(5) 对照程序逻辑,检查已设计出的测试用例的逻辑覆盖程度,如果没有达到要求的覆盖标准,应当再补充足够的测试用例。

(6) 如果程序的功能说明中含有输入条件的组合情况,则应在一开始就选用因果图法。

3.4 白盒测试

白盒测试也称作结构测试或逻辑驱动测试，使用白盒测试时知道产品的内部工作过程，通过测试来检测产品内部动作是否按照规格说明书的规定正常进行。白盒测试方法按照程序内部的结构测试程序，检验程序中的每条通路是否都能按预定要求正确工作，而不顾它的功能。白盒测试的主要方法有逻辑覆盖、基本路径测试等，它主要用于软件验证。

通常的程序结构覆盖有：

- 语句覆盖；
- 判断覆盖；
- 条件覆盖；
- 判断/条件覆盖；
- 条件组合覆盖；
- 路径覆盖。

语句覆盖是最常见也是最弱的逻辑覆盖准则，它要求设计若干个测试用例，使被测程序的每个语句都至少执行一次。判定覆盖或分支覆盖则要求设计若干个测试用例，使被测程序的每个判定的真、假分支都至少执行一次。当判定含有多个条件时，可以要求设计若干个测试用例，使被测程序的每个条件的真、假分支都至少执行一次，即条件覆盖，在考虑对程序路径进行全面检验时，即可使用条件覆盖准则。

虽然结构测试提供了评价测试的逻辑覆盖准则，但结构测试是不完全的。如果程序结构本身存在问题，比如程序逻辑错误或者遗漏了规格说明书中已规定的功能，那么，无论哪种结构测试，即使其覆盖率达到了百分之百，也是检查不出来的。因此，提高结构测试的覆盖率，可以增强对被测软件的信度，但并不能做到万无一失。

3.4.1 逻辑覆盖测试

白盒测试技术的常见方法之一就是覆盖测试，它利用程序的逻辑结构来设计相应的测试用例。测试人员要深入了解被测程序的逻辑结构特点，完全掌握源代码的流程，才能设计出恰当的用例。根据不同的测试要求，覆盖测试可以分为语句覆盖、判断覆盖、条件覆盖、判断/条件覆盖、条件组合覆盖和路径覆盖。

下面是一段简单的 C 语言程序，这里将其作为公共程序段来说明五种覆盖测试的各自特点。

```
If (x>100&& y>500) then
score = score + 1
If (x>= 1000|| z>5000) then
score = score + 5
```

其程序控制流图如图 3-8 所示。

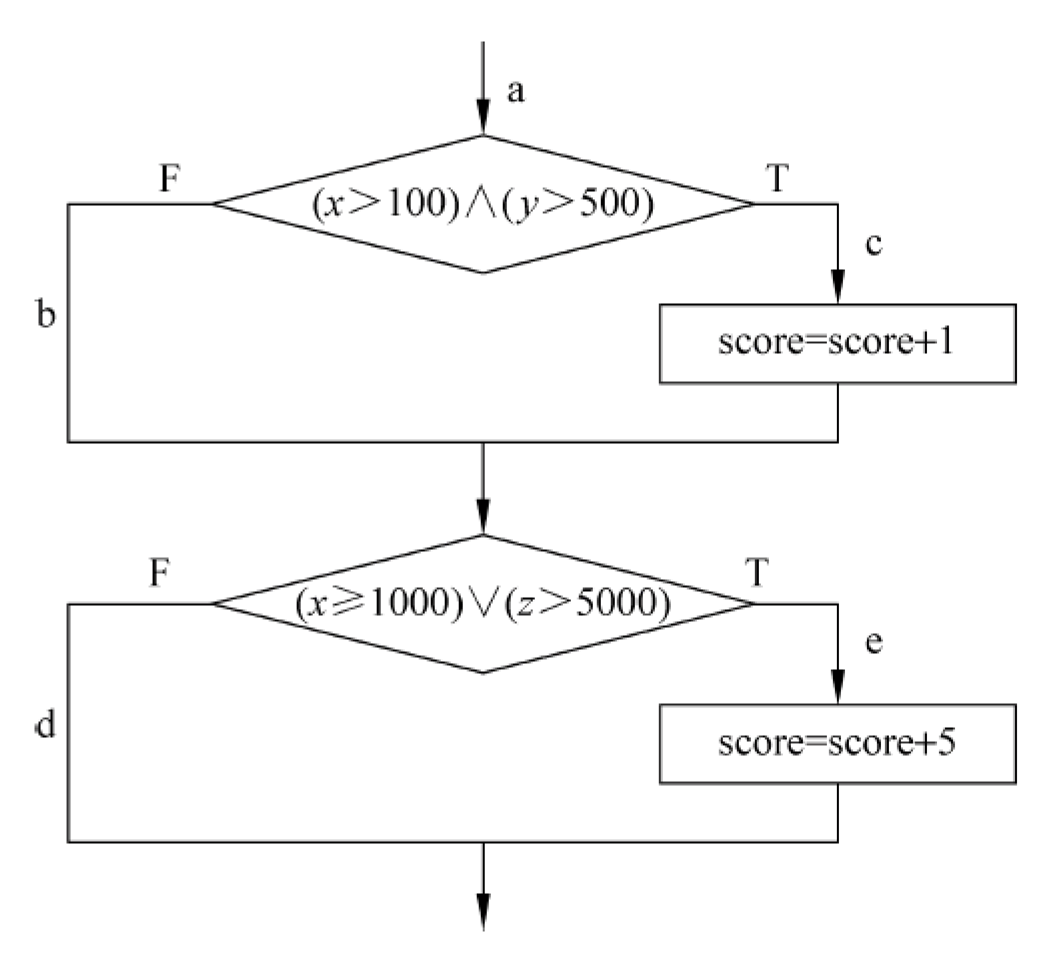


图 3-8　程序流程图

语句覆盖(Statement Coverage)是指设计若干个测试用例,程序运行时每个可执行语句至少被执行一次。在保证完成要求的情况下,测试用例的数目越少越好。

以下是针对公共程序段设计的两个测试用例,称为测试用例组 1。

```
Test Case 1: x = 2000, y = 600, z = 6000
Test Case 2: x = 900, y = 600, z = 5000
```

如表 3-14 所示,采用 Test Case 1 作为测试用例,则程序按路径 ace 顺序执行,程序中的 4 个语句都将执行一次,符合语句覆盖的要求。采用 Test Case 2 作为测试用例,则程序按路径 acd 顺序执行,程序中的语句 4 没有执行到,所以没有达到语句覆盖的要求。

表 3-14　测试用例组 1

测试用例	x,y,z	$(x>100)$and $(y>500)$	$(x>=1000)$or $(z>5000)$	执行路径
Test Case 1	2000,600,6000	True	True	ace
Test Case 2	900,600,5000	True	False	acd

从表面上看,语句覆盖用例测试了程序中的每一个语句行,好像对程序覆盖得很全面,但实际上语句覆盖测试是最弱的逻辑覆盖方法。例如,第一个判断的逻辑运算符“&&”错误写成“||”,或者第二个判断的逻辑运算符“||”错误地写成“&&”,这时如果采用 Test Case 1 测试用例是检验不出程序中的判断逻辑错误的。如果语句 3“If (x>=1000|| z>5000) then”错误写成“If (x>=1500|| z>5000) then”,Test Case 1 同样无法发现错误之处。

根据上述分析可知,语句覆盖测试只是表面上的覆盖程序流程,没有针对源程序各个语句间的内在关系,设计更为细致的测试用例。

判断覆盖(Branch Coverage)是指设计若干个测试用例,执行被测试程序时,使程序中每个判断条件的真值分支和假值分支至少执行一遍。在保证完成要求的情况下,测试用例的数目同样越少越好,判断覆盖又称为分支覆盖。

测试用例组 2：

```
Test Case 1: x = 2000, y = 600, z = 6000
Test Case 3: x = 50, y = 600, z = 2000
```

如表 3-15 所示，采用 Test Case 1 作为测试用例，程序按路径 ace 顺序执行；采用 Test Case 3 作为测试用例，程序按路径 abd 顺序执行。所以采用这一组测试用例，公共程序段的 4 个判断分支 b、c、d、e 都被覆盖到了。

表 3-15 测试用例组 2

测 试 用 例	x,y,z	$(x>100)$and$(y>500)$	$(x>=1000)$or$(z>5000)$	执 行 路 径
Test Case 1	2000,600,6000	True	True	ace
Test Case 3	50,600,2000	False	False	abd

测试用例组 3：

```
Test Case 4: x = 2000, y = 600, z = 2000
Test Case 5: x = 2000, y = 200, z = 6000
```

如表 3-16 所示，采用 Test Case 4 作为测试用例，程序沿着路径 acd 顺序执行；采用 Test Case 5 作为测试用例，则程序沿着路径 abe 顺序执行，显然采用这组测试用例同样可以满足判断覆盖。

表 3-16 测试用例组 3

测 试 用 例	x,y,z	$(x>100)$and$(y>500)$	$(x>=1000)$or$(z>5000)$	执 行 路 径
Test Case 4	2000,600,2000	True	False	acd
Test Case 5	2000,200,6000	False	True	abe

实际上，测试用例组 2 和测试用例组 3 不仅达到了判断覆盖要求，也同时满足了语句覆盖要求，某种程度上可以说判断覆盖测试要强于语句覆盖测试。但是，如果将第二个判断条件“(x>=1000)or (z>5000)”中的 z>5000 错误定义成 z 的其他限定范围，由于判断条件中的两个判断式是“或”的关系，其中一个判断式错误不影响结果，所以这两组测试用例是发现不了问题的，因此，应该用具有更强逻辑覆盖能力的覆盖测试方法来测试这种内部判断条件。

条件覆盖(Condition Coverage)是指设计若干个测试用例，执行被测试程序时，使程序中每个判断条件中的每个判断式的真值和假值至少执行一遍。

测试用例组 4：

```
Test Case 1: x = 2000, y = 600, z = 6000
Test Case 3: x = 50, y = 600, z = 2000
Test Case 5: x = 2000, y = 200, z = 6000
```

如表 3-17 所示，把前面设计过的测试用例挑选出 Test Case 1、Test Case 3、Test Case 5 组合成测试用例组 4，组中的 3 个测试用例覆盖了 4 个内部判断式的 8 种真假值情况，同时这组测试用例也实现了判断覆盖，但是并不可以说判断覆盖是条件覆盖的子集。

表 3-17 测试用例组 4

测试用例	x,y,z	$(x>100)$	$(y>500)$	$(x>=1000)$	$(z>5000)$	执行路径
Test Case 1	2000,600,6000	True	True	True	False	ace
Test Case 3	50,600,2000	False	True	False	False	abd
Test Case 5	2000,200,6000	True	False	True	True	abe

测试用例组 5：

```
Test Case 6: 50,600,6000
Test Case 7: 2000,200,1000
```

测试结果如表 3-18(a) 和表 3-18(b)所示，其中表 3-18(a)表示每个判断条件的每个判断式的真值和假值，表 3-18(b)表示每个判断条件的真值和假值。测试用例组 5 中的 2 个测试用例虽然覆盖了 4 个内部判断式的 8 种真假值情况，但是这组测试用例的执行路径是 abe，仅是覆盖了判断条件的 4 个真假分支中的 2 个，所以，需要设计一种能同时满足判断覆盖和条件覆盖的覆盖测试方法，即判断/条件覆盖测试。

表 3-18(a) 测试用例组 5

测试用例	x,y,z	$(x>100)$	$(y>500)$	$(x>=1000)$	$(z>5000)$	执行路径
Test Case 6	50,600,6000	False	True	False	True	abe
Test Case 7	2000,200,1000	True	False	True	False	abe

表 3-18(b) 测试用例组 5

测 试 用 例	x,y,z	$(x>100)$and $(y>500)$	$(x>=1000)$or $(z>5000)$	执 行 路 径
Test Case 6	50,600,6000	False	True	abe
Test Case 7	2000,200,1000	False	True	abe

判断/条件覆盖是指设计若干个测试用例，使执行被测试程序时，程序中每个判断条件的真假值分支至少执行一遍，并且每个判断条件的内部判断式的真假值分支也要被执行一遍。

测试用例组 6：

```
Test Case 1: x = 2000, y = 600, z = 2000
Test Case 6: x = 2000, y = 200, z = 6000
Test Case 7: x = 2000, y = 600, z = 2000
Test Case 8: x = 50, y = 200, z = 2000
```

测试结果如表 3-19(a) 和表 3-19(b)所示，其中表 3-19(a)表示每个判断条件的每个判断式的真值和假值，表 3-19(b)表示每个判断条件的真值和假值。测试用例组 6 虽然满足了判断覆盖和条件覆盖，但是没有对每个判断条件的内部判断式的所有真假值组合进行测试。条件组合判断是必要的，因为条件判断语句中的“与”和“或”，即“&&”和“||”，会使内部判断式之间产生抑制作用。例如，C=A && B 中，如果 A 为假值，那么 C 就为假值，测试程序就不检测 B 了，B 的正确与否就无法测试。同样，C=A || B 中，如果 A 为真值，那么 C 就为真值，测试程序也不检测 B 了，B 的正确与否也就无法测试。

条件组合覆盖则是指设计若干个测试用例,使执行被测试程序时,程序中每个判断条件的内部判断式的各种真假组合可能都至少执行一遍。可见,满足条件组合覆盖的测试用例组一定满足判断覆盖、条件覆盖和判断/条件覆盖。

测试用例组 7:

```
Test Case 1: x = 2000, y = 600, z = 2000
Test Case 6: x = 2000, y = 200, z = 6000
Test Case 7: x = 2000, y = 600, z = 2000
Test Case 8: x = 50, y = 200, z = 2000
```

测试结果如表 3-20(a) 和表 3-20(b)所示,表 3-20(a)表示每个判断条件的每个判断式的真值和假值,表 3-20(b)表示每个判断条件的真值和假值。测试用例组 7 虽然满足了判断覆盖、条件覆盖以及判断/条件覆盖,但是并没有覆盖程序控制流图中全部的 4 条路径(ace,abe,abe,abd),只覆盖了其中 3 条路径(ace,abe,abd)。软件测试的目的是尽可能地发现所有软件缺陷,因此程序中的每一条路径都应该进行相应的覆盖测试,从而保证程序中的每一个特定路径方案都能顺利运行。能够达到这样要求的是路径覆盖测试,在下一节将进行介绍。

表 3-19(a) 测试用例组 6

测试用例	x,y,z	$(x>100)$	$(y>500)$	$(x>=1000)$	$(z>5000)$	执行路径
Test Case 1	2000,600,6000	True	True	True	True	ace
Test Case 8	50,200,2000	False	False	False	False	abd

表 3-19(b) 测试用例组 6

测 试 用 例	x,y,z	$(x>100)$ and $(y>500)$	$(x>=1000)$ or $(z>5000)$	执 行 路 径
Test Case 1	2000,600,6000	True	True	ace
Test Case 8	50,200,2000	False	False	abd

表 3-20(a) 测试用例组 7

测试用例	x,y,z	$(x>100)$	$(y>500)$	$(x>=1000)$	$(z>5000)$	执行路径
Test Case 1	2000,600,6000	True	True	True	True	ace
Test Case 6	50,600,6000	False	True	False	True	abe
Test Case 7	2000,200,1000	True	False	True	False	abe
Test Case 8	50,200,2000	False	False	False	False	abd

表 3-20(b) 测试用例组 7

测 试 用 例	x,y,z	$(x>100)$ and $(y>500)$	$(x>=1000)$ or $(z>5000)$	执 行 路 径
Test Case 1	2000,600,6000	True	True	ace
Test Case 6	50,600,6000	False	True	abe
Test Case 7	2000,200,1000	False	True	abe
Test Case 8	50,200,2000	False	False	abd

应该注意的是，上面 6 种覆盖测试方法所引用的公共程序只有短短 4 行，是一段非常简单的示例代码，然而在实际程序测试中，即便一个简短的程序，其路径数目也可能是一个庞大的数字。要对其实现路径覆盖测试是很难的。所以，路径覆盖测试是相对的，要尽可能把路径数压缩到一个可承受范围。

当然，即便对某个简短的程序段做到了路径覆盖测试，也不能保证源代码不存在其他软件问题了。多手段的软件测试是必要的，它们之间是相辅相成的。没有一个测试方法能够找尽所有软件缺陷，只能说是尽可能多地查找软件缺陷。

3.4.2　路径分析测试

路径覆盖是白盒测试最为典型的问题，但大多数情况下实现路径覆盖几乎是不可能的，此时可进行着眼于路径分析的测试，称为路径分析测试。完成路径测试的理想情况是做到路径覆盖。独立路径选择和 Z 路径覆盖是两种常见的路径覆盖方法。

1. 控制流图

白盒测试是针对软件产品内部逻辑结构进行测试的，测试人员必须对测试的软件有深入的理解，包括其内部结构、各单元部分及它们之间的内在联系，还有程序运行原理等。因而这是一项庞大并且复杂的工作。为了更加突出程序的内部结构，便于测试人员理解源代码，可以对程序流程图进行简化，生成控制流图(Control Flow Graph)。简化后的控制流图将由节点和控制边组成。

1) 控制流图的特点

控制流图有以下几个特点：

① 具有唯一入口节点，即源节点，表示程序段的开始语句；

② 具有唯一出口节点，即汇节点，表示程序段的结束语句；

③ 节点由带有标号的圆圈表示，表示一个或多个无分支的源程序语句；

④ 控制边由带箭头的直线或弧表示，代表控制流的方向。

常见的控制流图如图 3-9 所示。

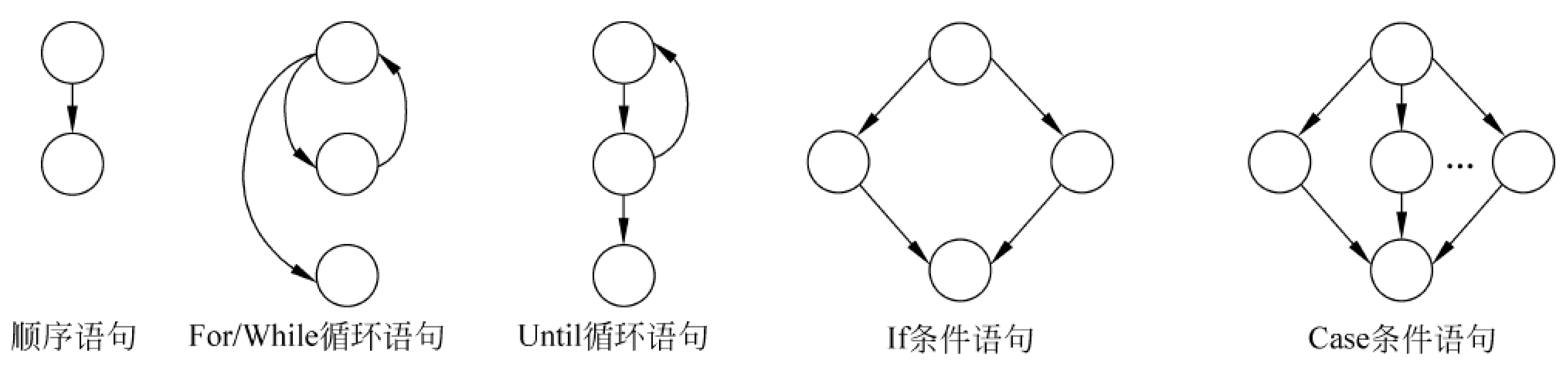


图 3-9　常见的控制流图

包含条件的节点被称为判断节点，由判断节点发出的边必须终止于某一个节点。

2) 程序环路复杂性

程序的环路复杂性是一种描述程序逻辑复杂度的标准，该标准运用基本路径方法，给出了程序基本路径集中的独立路径条数，这是确保程序中每个可执行语句至少执行一次所必需的测试用例数目的上界。

给定一个控制流图 G，设其环形复杂度为 V(G)，这里介绍 3 种常见的求解 V(G)的计算方法。

① V(G)＝E－N＋2，其中 E 是控制流图 G 中边的数量，N 是控制流图中节点的数目。

② V(G)＝P＋1，其中 P 是控制流图 G 中判断节点的数目。

③ V(G)＝A，其中 A 是控制流图 G 中区域的数目。由边和节点围成的区域叫做区域，当在控制流图中计算区域的数目时，控制流图外的区域也应记为一个区域。

2. 独立路径测试

对于一个较为复杂的程序要做到完全的路径覆盖测试是不可能实现的，既然路径覆盖测试无法达到，那么可以对某个程序的所有独立路径进行测试，从而可以认为已经检验了程序的每一条语句，即达到了语句覆盖，这种测试方法就是独立路径测试方法。从控制流图来看，一条新独立路径应至少包含一条在其他独立路径中从未有过的边，路径可以用控制流图中的节点序列来表示。

例如，在如图 3-10 的控制流图中，一组独立的路径如下。

path1：1→11

path2：1→2→3→4→5→10→1→11

path3：1→2→3→6→8→9→10→1→11

path4：1→2→3→6→7→9→10→1→11

路径 path1、path2、path3、path4 组成了控制流图的一个基本路径集。

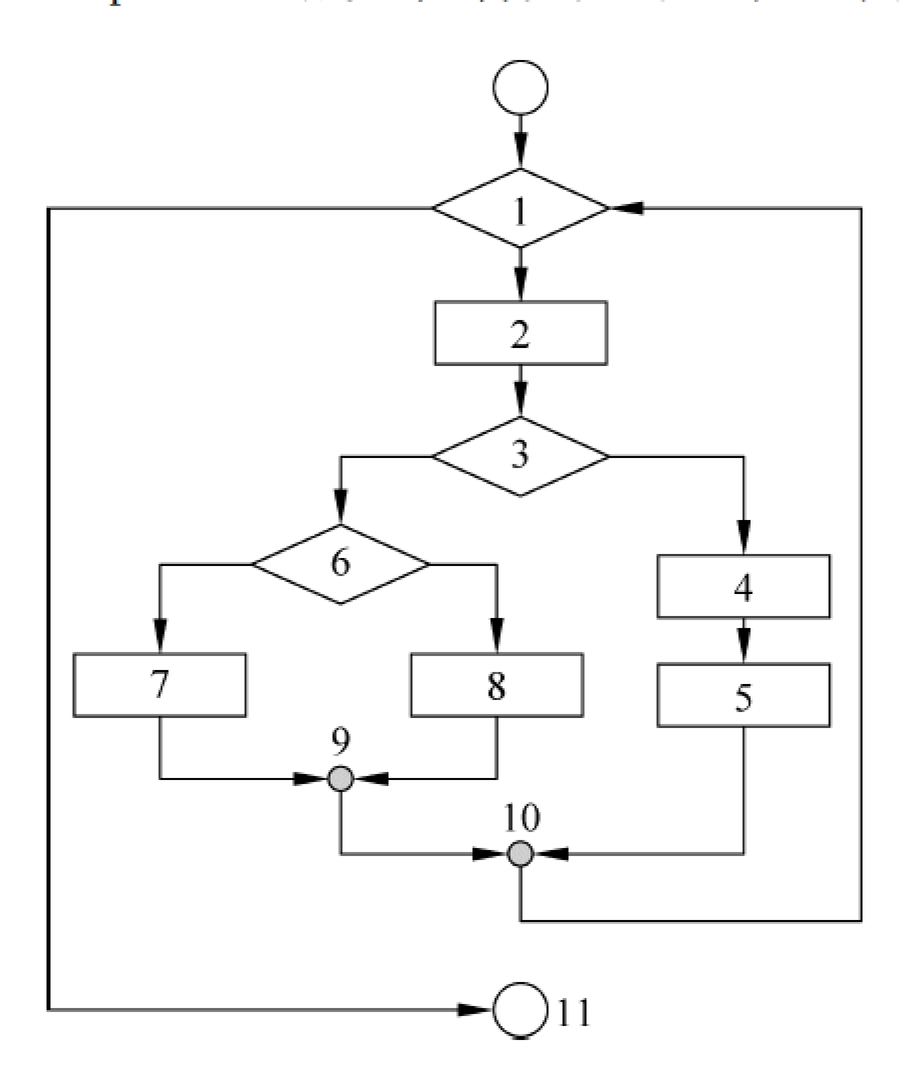


图 3-10 控制流图示例

白盒测试可以设计成基本路径集的执行过程，通常，基本路径集并不唯一确定，独立路径测试的步骤包括 3 个方面：导出程序控制流图，求程序环形复杂度，设计测试用例(Test Case)。

下面通过一个 C 语言程序实例来具体说明独立路径测试的设计流程，这段程序的作用是统计一行字符中有多少个单词，单词之间用空格分隔开。

```
1   main ()
2   {
3    int num1 = 0, num2 = 0, score = 100;
4   int i;
5    char str;
6    scanf ("%d, %c\n", &i, &str);
7    while (i<5)
8    {
9        if (str = 'T')
10         num1++;
11       else if (str = 'F')
12       {
13         score = score - 10;
14         num2 ++;
15        }
16       i++;
17   }
18   printf ("num1 = %d, num2 = %d, score = %d\n", num1, num2, score);
19  }
```

根据源代码可以导出程序的控制流图，如图 3-11 所示。每个圆圈代表控制流图的节点，表示一个或多个语句，圆圈中的数字对应程序中某一行的编号，箭头代表边的方向，即控制流方向。

然后根据程序环形复杂度的计算公式，求出程序路径集合中的独立路径数目。

公式 1：V(G)＝10－8＋2，其中 10 是控制流图 G 中边的数量，8 是控制流图中节点的数目。

公式 2：V(G)＝3＋1，其中 3 是控制流图 G 中判断节点的数目。

公式 3：V(G)＝4，其中 4 是控制流图 G 中区域的数目。

因此，控制流图 G 的环形复杂度是 4，就是说至少需要 4 条独立路径组成基本路径集合，并由此得到能够覆盖所有程序语句的测试用例。

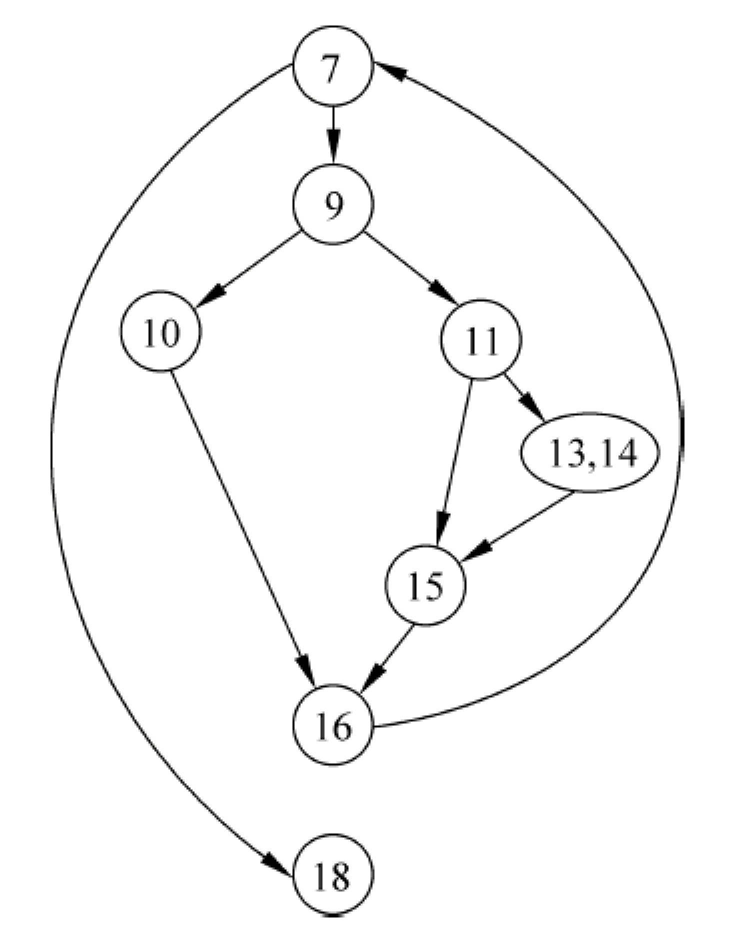


图 3-11　控制流图

下面就来设计测试用例。根据上面环形复杂度的计算结果，源程序的基本路径集合中有 4 条独立路径：

path1：7→18

path2：7→9→10→16→7→18

path3：7→9→11→15→16→7→18

path4：7→9→11→13→14→15→16→7→18

根据上述 4 条独立路径，设计了测试用例组 8，如表 3-21 所示。将测试用例组 8 中的 4 个测试用例作为程序输入数据，能够遍历这 4 条独立路径，源程序中的循环体分别将执行零次或一次。

表 3-21　测试用例组 8

测试用例	输　　入		期 望 输 出			执行路径
	i	str	num1	num2	score	
Test Case 1	5	‘T’	0	0	100	路径 1
Test Case 2	4	‘T’	1	0	100	路径 2
Test Case 3	4	‘A’	0	0	100	路径 3
Test Case 4	4	‘F’	0	1	90	路径 4

注意：如果程序中的条件判断表达式是由一个或多个逻辑运算符(or,and,not)连接的复合条件表达式,需要变换为一系列只有单个条件的嵌套的判断。例如：

```
1  if (a or b)
2  then
3    procedure x
4  else
5    procedure y;
6  …
```

对应的控制流图如图 3-12 所示,程序行 1 的 a、b 都是独立的判断节点,还有程序行 4 也是判断节点,所以共计 3 个判断节点。图 3-12 的环形复杂度因此为 V(G)＝3＋1。

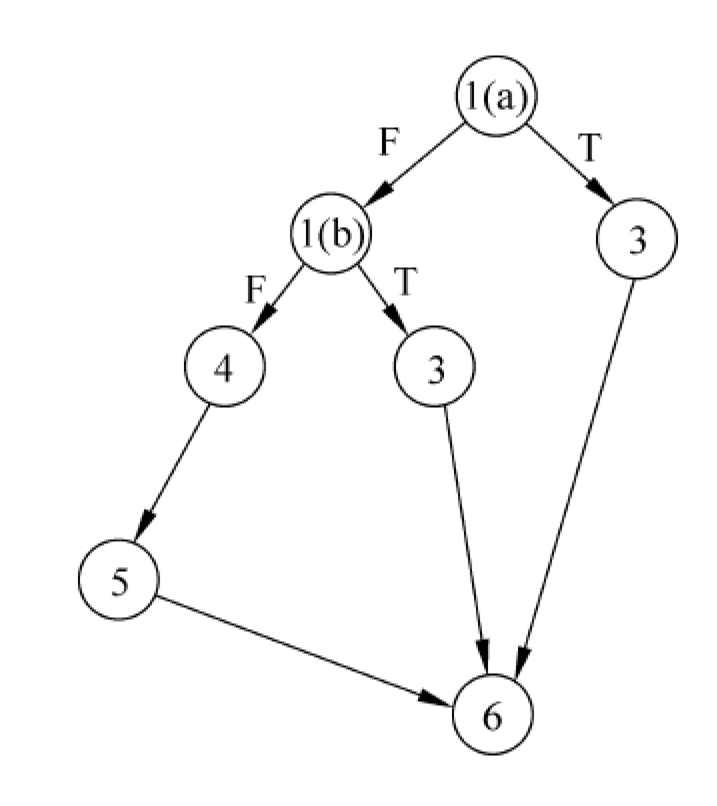


图 3-12　程序控制流图

3. Z 路径覆盖测试

和独立路径选择一样,Z 路径覆盖也是一种常见的路径覆盖方法,可以说 Z 路径覆盖是路径覆盖的一种变体。对于语句较少的简单程序,路径覆盖是具有可行性的,但是对于源代码很多的复杂程序,或者对于含有较多条件语句和较多循环体的程序来说,需要测试的路径数目会成倍增长,达到一个巨大数字,以至于无法实现路径覆盖。

为了解决这一问题,必须舍弃一些不重要的因素,简化循环结构,从而极大地减少路径的数量,使得覆盖这些有限的路径成为可能。采用简化循环方法的路径覆盖就是 Z 路径覆盖。

所谓简化循环就是减少循环的次数。这里不考虑循环体的形式和复杂度如何,也不考虑循环体实际上需要执行多少次,只考虑通过循环体零次和一次这两种情况。零次循环即是指跳过循环体,从循环体的入口直接到循环体的出口。通过一次循环体则是为了检查循环初始值。

根据简化循环的思路,循环要么执行,要么跳过,这和判定分支的效果是一样的,可见,简化循环就是将循环结构转变成选择结构。

4. 白盒测试的优缺点

(1) 白盒测试的优点

① 白盒测试方法深入到了程序内部,测试粒度到达某个模块、某个函数甚至某条语句,

能从程序具体实现的角度发现问题；

② 白盒测试方法是对黑盒测试方法的最有力补充，只有将二者结合才能将软件测试工作做到相对到位。

（2）白盒测试的缺点

① 白盒测试使测试人员集中关注程序是否正确执行，却很难同时让测试人员考虑是否完全满足设计说明书、需求说明书或者用户实际需求，也较难查出程序中遗漏的路径；

② 白盒测试方法的高覆盖率要求，使得测试工作量大，远远超过黑盒测试的工作量；

③ 需要测试人员用尽量短的时间理解开发人员编写的代码；

④ 需要测试人员读懂代码（思维进入程序）后，还能站在一定高度（思维跳出程序）设计测试用例和开展测试工作，这对测试人员要求太高。

3.5　本章小结

本章从不同角度对软件测试方法加以划分，重点介绍了黑盒测试和白盒测试。黑盒测试是一种确认技术，目的是确认“设计的系统是否正确”，黑盒测试是以用户的观点，从输入数据与输出数据的对应关系，也就是根据程序外部特性进行的测试，而不考虑程序内部结构及工作情况。白盒测试方法深入到了程序内部，能从程序具体实现的角度发现问题。

习题 3

1. 简述软件测试技术从不同角度加以划分的多种方法。
2. 简述静态测试和动态测试的区别。
3. 举例说明黑盒测试的几种测试方法。
4. 举例说明覆盖测试的几种测试方法。
5. 简述白盒测试的相关方法。
6. 比较阐述黑盒测试和白盒测试的优缺点。

第4章 软件测试过程

当软件设计及实现工作完成以后，程序就会被提交给测试组，测试负责人开始组织测试。一般可按下列方式组织测试：首先测试人员要仔细阅读有关资料，包括规格说明、设计文档、使用说明书及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试用例，做好测试前的准备工作。

为了保证测试的质量，通常将测试过程分成几个阶段，即：单元测试、集成测试、确认测试和系统测试等。本章将分别介绍这几个阶段的主要任务和测试方法。

4.1 软件测试过程概述

软件测试过程用于定义软件测试的流程和方法。众所周知，开发过程的质量决定了软件的质量，同样，测试过程的质量将直接影响测试结果的准确性和有效性。软件测试过程和软件开发过程一样，都遵循软件工程原理和管理学原理。

完整的软件测试流程包括制定测试计划、测试设计、测试开发、测试执行和测试评估几个部分。

(1) 制定测试计划。即根据用户需求报告中关于功能要求和性能指标的规格说明书，定义相应的测试需求报告，使得随后所有的测试工作都围绕着测试需求来进行。同时，适当选择测试内容，合理安排测试人员、测试时间及测试资源等。

(2) 测试设计。测试设计是指将测试计划阶段制定的测试需求分解、细化为若干个可执行的测试过程，并为每个测试过程选择适当的测试用例，保证测试结果的有效性。

(3) 测试开发。建立可重复使用的自动测试过程。

(4) 测试执行。执行测试开发阶段建立的自动测试过程，并对所发现的缺陷进行跟踪管理。测试执行一般由单元测试、集成测试、确认测试、系统测试、验收测试以及回归测试等步骤组成。

(5) 测试评估。结合量化的测试覆盖域及缺陷跟踪报告，对应用软件的质量和开发团队的工作进度及工作效率进行综合评价。

在上述测试流程中，测试执行按以下步骤进行：单元测试、集成测试、确认测试和验收测试，如图 4-1 所示。现介绍如下。

- 单元测试(Unit Testing)，此阶段集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能，确保其能正常工作。

- 集成测试(Integration Testing)是把已进行过单元测试的模块组装起来进行测试，目的在于检验与软件设计相关的程序结构问题。
- 确认测试(Confirmation Testing)是检查已实现的软件是否满足了需求规格说明中确定了的各种功能和性能需求，以及软件配置是否完全和正确。
- 系统测试(System Testing)是把已经经过确认的软件纳入实际运行环境中，与其他系统成分(如数据库、硬件和操作人员)组合在一起进行测试。
- 验收测试(Acceptance Testing)是检验软件产品的最后一道工序，主要突出用户的作用，同时软件开发人员也应在一定的程度上参与。

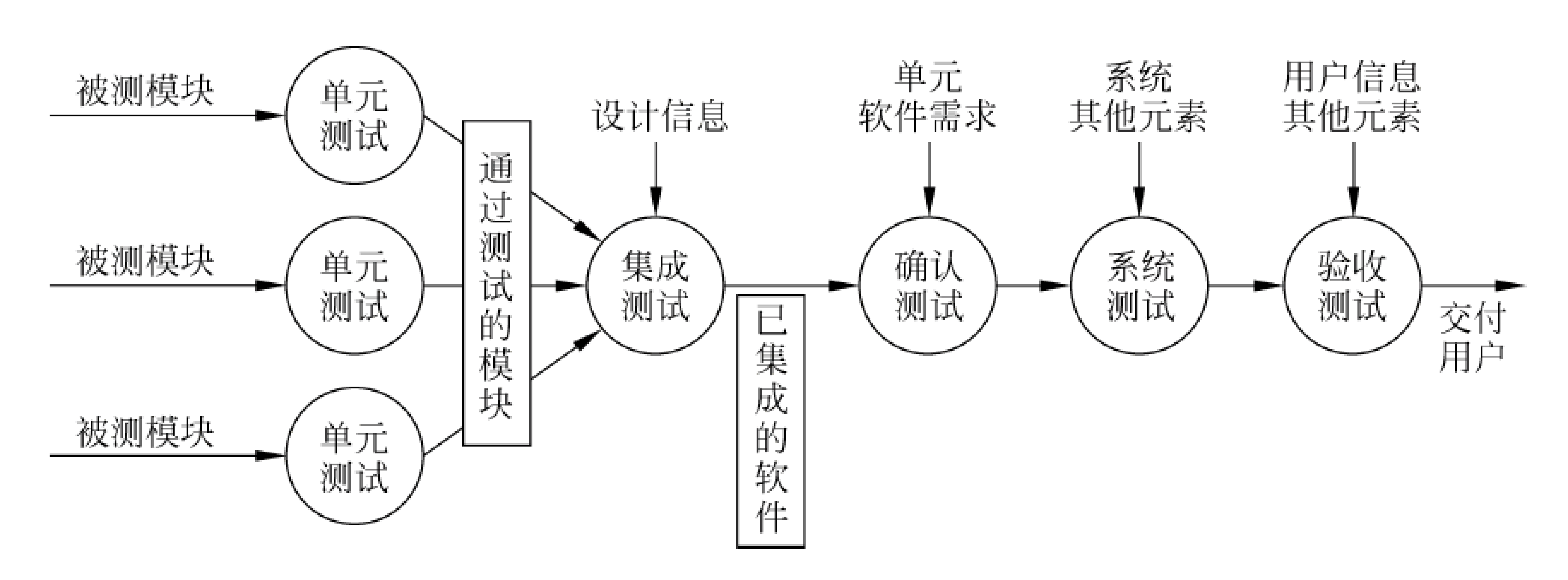


图 4-1 软件测试执行过程

软件测试执行各阶段的输入和输出标准如表 4-1 所示。

表 4-1 软件测试各阶段的输入和输出标准

阶段	输入	要求	输出
单元测试	源程序、编程规范、产品规格设计说明书和详细的程序设计文档	遵守规范、模块的高内聚性、功能实现的一致性和正确性	缺陷报告、跟踪报告；完善的测试用例、测试计划
集成测试	通过单元测试的模块或组件、编程规范、集成测试规格说明和程序设计文档、系统设计文档	接口定义清楚且正确、模块或组件一起工作正常、能集成为完整的系统	缺陷报告、跟踪报告；完善的测试用例、测试计划；集成测试分析报告；集成后的系统
确认测试	代码软件包(含文档)，功能详细设计说明书；测试计划和用例	模块集成、功能的正确性和适用性	缺陷报告、代码完成状态报告、功能验证测试报告
系统测试	修改后的软件包、测试环境、系统测试用例和测试计划	系统能正常地、有效地运行，包括性能、可靠性、安全性、兼容性等	缺陷报告、系统性能分析报告、缺陷状态报告、阶段性测试报告
验收测试	产品规格设计说明、预发布的软件包、确认测试用例	向用户标明系统能够按照预定要求那样工作，使系统最终可以正式发布或向用户提供服务。用户要参与验收测试，包括 α 测试和 β 测试	用户验收报告、缺陷报告审查、版本审查、最终测试报告

4.2 单元测试

单元测试又称模块测试，主要测试软件设计的最小单位(模块或组件)在语法、格式或逻辑等方面的差错以及是否符合功能要求。这个阶段更多关注程序实现的细节，需要从程序的内部结构出发设计测试用例。单元测试也是软件测试过程中最早期的测试活动。

在程序员完成编码，并且代码通过编译后一般就可以进行单元测试了。由于国内的单元测试往往不正规，因此单元测试多由开发人员自己来完成，一般采用交叉测试方法。

4.2.1 单元测试的主要任务

单元测试的主要依据是源程序代码和详细设计说明书，测试者需要了解该模块的I/O条件和模块的逻辑结构。单元测试主要采用白盒测试的测试用例，辅以黑盒测试的测试用例，使之对任何合理的和不合理的输入都能鉴别和响应。单元测试主要有以下5个任务。

1. 模块接口测试

即对通过被测试模块的数据流进行测试，检查进出模块的数据是否正确。为此，必须对全部的模块接口，包括参数表、调用子模块的参数、全程数据、文件输入/输出操作等进行测试。

(1) 测试项目包括以下内容：

① 调用其他模块时，所传送的实际参数个数与被调用模块的形式参数的个数是否相同；所传送的实际参数与被调用模块的形式参数的类型是否匹配；所传送的实际参数与被调用模块的形式参数的单位是否一致。

② 调用内部函数时，参数的个数、属性和次序是否正确。

③ 在模块有多个入口的情况下，是否引用与当前入口无关的参数。

④ 是否修改了只读型参数。

⑤ 全局变量是否在所有引用它们的模块中都有相同的定义。

(2) 如果模块内包括外部I/O，还应该考虑下列因素：

① 文件属性是否正确。

② OPEN与CLOSE语句是否正确。

③ 缓冲区容量与记录长度是否匹配。

④ 在进行读/写操作之前是否打开了文件。

⑤ 在结束文件处理时是否关闭了文件。

⑥ 正文书写/输入有无错误。

⑦ I/O错误是否检查并做了处理。

2. 局部数据结构测试

局部数据结构测试应注意以下几类错误：不一致或不正确的数据类型说明；使用尚未

赋值或初始化的变量；错误的初始值或默认值；变量名拼写或书写错误；上溢、下溢或者地址错误等方面的问题。此外，应注意全局数据对模块的影响。

在模块工作过程中，必须测试模块内部的数据能否保持完整性，包括内部数据的内容、形式及相互关系不发生错误。

3. 路径测试

路径测试是对模块中重要的执行路径进行测试，其中对基本执行路径和循环进行测试往往可以发现大量路径错误。

路径测试的测试用例必须能够发现由于计算错误而产生的错误。常见的错误如下：误解的或不正确的算术优先级；混合模式的运算错误；错误的初始化；精确度不够；表达式的不正确符号表示。

此外，还必须能够发现不正确的判定或不正常的控制流而产生的错误。常见的错误如下：不同数据类型的比较错误；不正确的逻辑操作或优先级；应当相等的地方由于精确度的错误而不能相等；不正确的判定或不正确的变量；不正确的或不存在的循环终止；当遇到分支循环时不能退出；不适当地修改循环变量。

4. 错误处理测试

检查模块的错误处理功能是否包含错误或缺陷，处理设施是否有效。例如，是否可拒绝不合理的输入，出错的描述是否难以理解，出错的描述是否不能够对错误定位，显示的错误与实际的错误是否相符，对错误条件的处理是否正确，在对错误处理之前错误条件是否已经引起系统的干预等。

5. 边界条件测试

边界条件测试是单元测试的最后一步，必须采用边界值分析方法来设计测试用例，注意数据流、控制流中刚好等于、大于或小于确定的边界值时，模块是否能够正常工作。

在边界条件测试中，应设计测试用例检查以下情况：

(1) 在 n 次循环的第 0 次、1 次、n 次是否有错误。

(2) 运算或判断中取最大值、最小值时是否有错误。

(3) 数据流、控制流中刚好等于、大于、小于确定的比较值时是否出现错误。

在单元测试中，如果对模块运行时间有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。

4.2.2　单元测试的执行过程

通常单元测试在编码阶段进行，在源程序代码编制完成并经过评审和验证，确认没有语法错误之后，就可以开始设计单元测试的测试用例。利用设计文档，设计可以验证程序功能、找出程序错误的多个测试用例。对于每一组输入，应有预期的正确结果。

模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，因此可使用一些辅助模块去模拟与被测模块相关的其他模块。辅助模块分为以下两种。

- 驱动模块(Drive)：用来模拟被测模块的上一级模块，相当于被测模块的主程序，用

于接收测试数据，并把这些数据传送给被测模块，启动被测模块，最后输出实测结果。

• 桩模块(Stub)：用来模拟被测模块工作过程中所调用的模块。桩模块由被测模块调用，一般只进行很少的数据处理(例如打印入口和返回)，以便检验被测模块与其下级模块的接口。桩模块不需要把子模块的所有功能都带进来，但不允许什么事情也不做。

被测模块和与它相关的驱动模块及桩模块共同构成了一个“测试环境”，如图 4-2 所示。

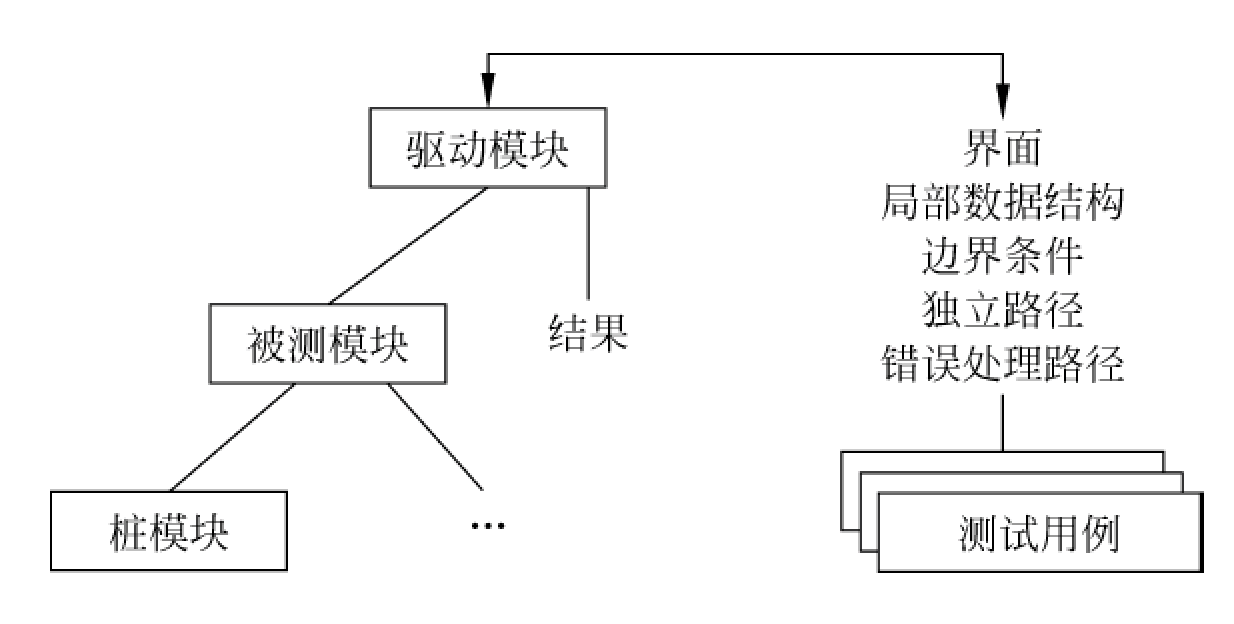


图 4-2 单元测试的测试环境

4.3 集成测试

按照软件设计要求，将经过单元测试的模块连接起来，组成所规定的软件系统的过程称为“集成”。实践表明，一些模块虽然能够单独地工作，但并不能保证连接起来也能正常工作，程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。集成测试就是针对这个过程，按模块之间的依赖接口关系图进行的测试。图 4-3 给出了软件分层结构的示例图。

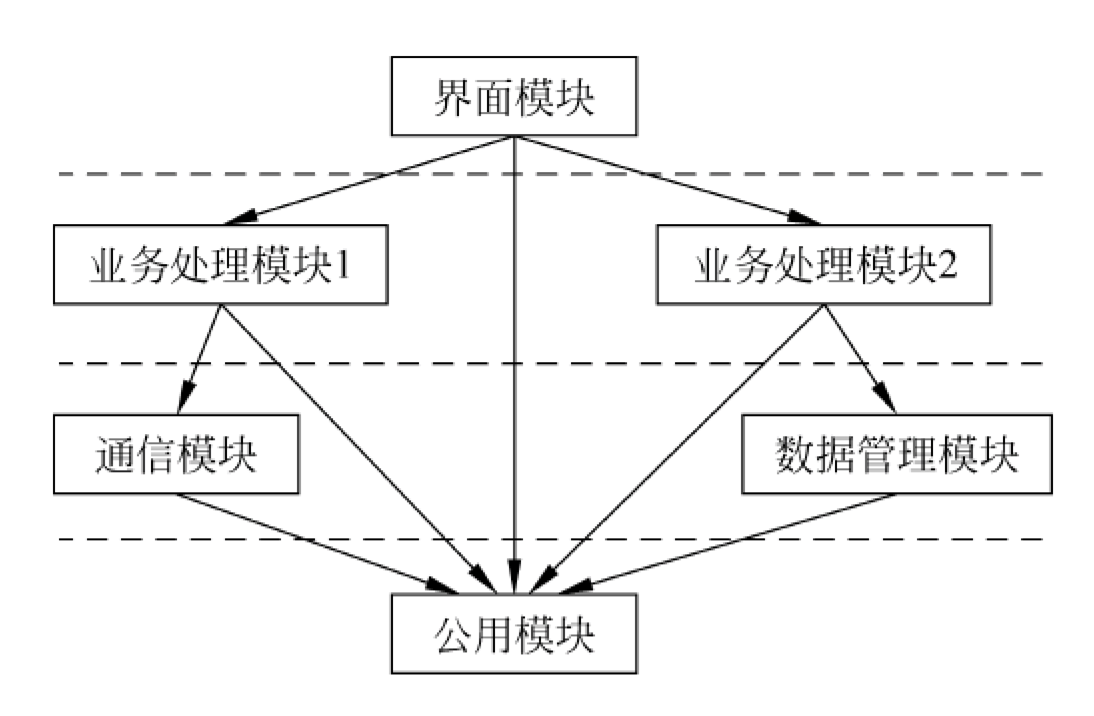


图 4-3 软件分层结构的示意图

由于集成测试不是在真实环境下进行，而是在开发环境或是一个独立的测试环境下进行的，因此集成测试所需人员一般从开发组中选出，在开发组长的监督下进行。开发组长负责保证在合理的质量控制和监督下使用合适的测试技术执行充分的集成测试。在集成测试过程中，由一个独立测试观察员来监控测试工作。

4.3.1　集成测试的主要任务

集成测试是组装软件的系统测试技术之一，其主要目标是在按设计要求把通过单元测试的各个模块组装在一起之后，确保软件系统符合实际软件结构，发现与接口有关的各种错误。

1. 集成测试主要适应于如下几种软件系统

（1）对软件质量要求较高的软件系统，如航天软件、电信软件、系统底层软件等。
（2）使用范围比较广、用户群数量较大的软件。
（3）使用类似 C/C++带有指针的程序语言开发的软件。
（4）类库、中间件等产品。

2. 集成测试的主要任务是解决以下 5 个问题

（1）将各模块连接起来，检查模块相互调用时数据经过接口是否丢失。
（2）将各个子功能组合起来，检查能否达到预期要求的各项功能。
（3）一个模块的功能是否会对另一个模块的功能产生不利的影响。
（4）全局数据结构是否有问题，会不会被异常修改。
（5）单个模块的误差积累起来是否被放大，以致达到不可接受的程度。

3. 集成测试方法

集成测试主要测试软件的结构问题，因此测试建立在模块接口上，多为黑盒测试，适当辅以白盒测试。

（1）执行集成测试应遵循如下步骤：
① 确认组成一个完整系统的模块之间的关系。
② 评审模块之间的交互和通信需求，确认模块间的接口。
③ 生成一套测试用例。
④ 采用增量式测试，依次将模块加入系统并测试，这个过程按逻辑/功能顺序重复进行。

（2）集成测试过程中要注意关键模块的测试，关键模块一般具有下述一个或多个特征：
① 同时对应几项需求功能。
② 具有高层控制功能。
③ 复杂、易出错。
④ 有特殊的性能要求。

集成测试的主要目的是验证组成软件系统的各模块的接口和交互作用，因此集成测试对数据的要求从难度和内容上不是很高。集成测试一般不使用真实数据，可以使用一部分代表性的测试数据。在创建测试数据时，应保证数据充分测试软件系统的边界条件。

4.3.2　集成测试的方法

选择什么样的方法把模块组装起来形成一个可运行的系统，直接影响到测试成本、测试

计划、测试用例的设计、测试工具的选择等。通常有以下两种集成测试方法。

1. 非增量式集成测试方法

非增量式集成测试方法采用一步到位的方法来进行测试，即对所有模块进行个别的单元测试后，按程序结构图将各个模块连接起来，把连接后的程序当作一个整体进行测试。

2. 增量式集成测试方法

增量式集成测试方法的集成是逐步实现的，集成测试也是逐步完成的，也可以说它将单元测试与集成测试结合起来进行。首先对一个个模块进行模块测试，然后将这些模块逐步组装成较大的系统，在集成的过程中边连接边测试，以发现连接过程中产生的问题，通过增量逐步组装成为要求的软件系统。

当对两个以上模块进行集成时，不可能忽视它们和周围模块的相互关系。为模拟这种联系，需设置若干辅助测试模块，也就是连接被测试模块的程序段。和单元测试阶段一样，辅助模块通常有驱动模块和桩模块两种。

增量式集成测试可以按照不同的次序实施，通常有下面 3 种方法。

（1）自顶向下增量式测试

自顶向下增量式测试按结构图自上而下进行逐步集成和逐步测试。模块集成的顺序是首先集成主控模块（主程序），然后按照软件控制层次结构向下进行集成。自顶向下的集成方式可以采用深度优先策略和广度优先策略两种，如图 4-4 所示。

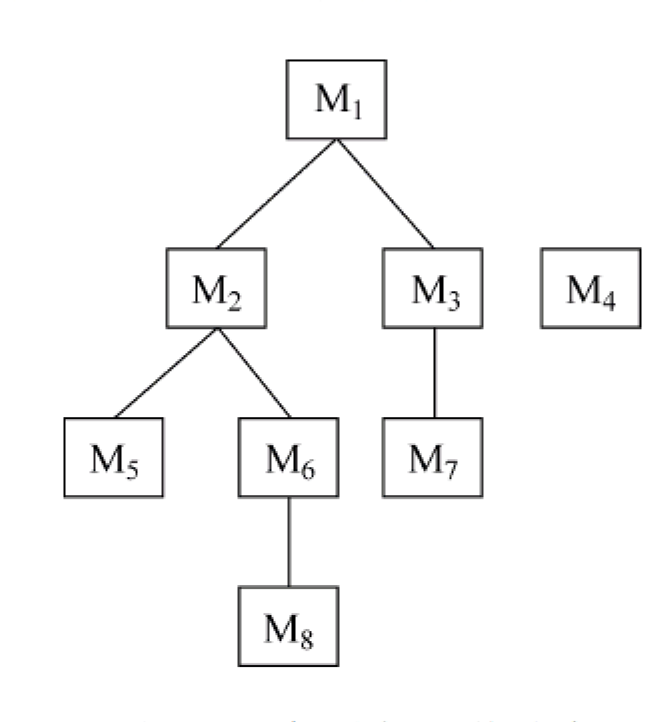


图 4-4 自顶向下增量式测试示意图

图 4-4 所示的集成顺序采用了广度优先策略：$M_1 \to M_2 \to M_3 \to M_4 \to M_5 \to M_6 \to M_7 \to M_8$。若采用深度优先则顺序为：$M_1 \to M_2 \to M_5 \to M_6 \to M_8 \to M_3 \to M_7 \to M_4$。

① 自顶向下增量式测试方法由下列步骤实现。

步骤 1：以主模块为所测试模块兼驱动模块，而所有直属于主模块的下属模块全部用桩模块替换，并对主模块进行测试。

步骤 2：采用深度优先或广度优先方法，用实际模块替换相应桩模块，再用桩模块代替它们的直接下属模块，从而与已经测试的模块或子系统组装成新的子系统。

步骤 3：进行回归测试排除组装过程中的错误可能性。

步骤 4：判断是否所有的模块都已经组装到了系统中。如果是，则结束测试，否则转到步骤 2 执行。

② 自顶向下增量式测试方法的优点如下：

- 在测试过程中可较早地验证主要的控制点。
- 功能性的模块测试可以较早地得到证实。
- 最多只需要一个驱动模块就可进行测试。
- 支持缺陷故障隔离。

③ 自顶向下增量式测试方法具有如下缺点：

- 随着底层模块的不断增加，可能导致底层模块的测试不充分，特别是被重用的模块。
- 由于每次组装都必须提供桩模块，会使得桩模块的数目急剧增加，从而维护桩模块的成本也快速上升。

因此，该方法主要适用于大部分采用结构化编程方法，而且软件的结构相对比较简单的情况。

(2) 自底向上增量式测试

自底向上增量式测试是从“原子”模块(即软件结构中最底层的模块)开始，按结构图自下而上逐步进行集成和测试。图 4-5 表示采用自底向上增量式测试的过程。

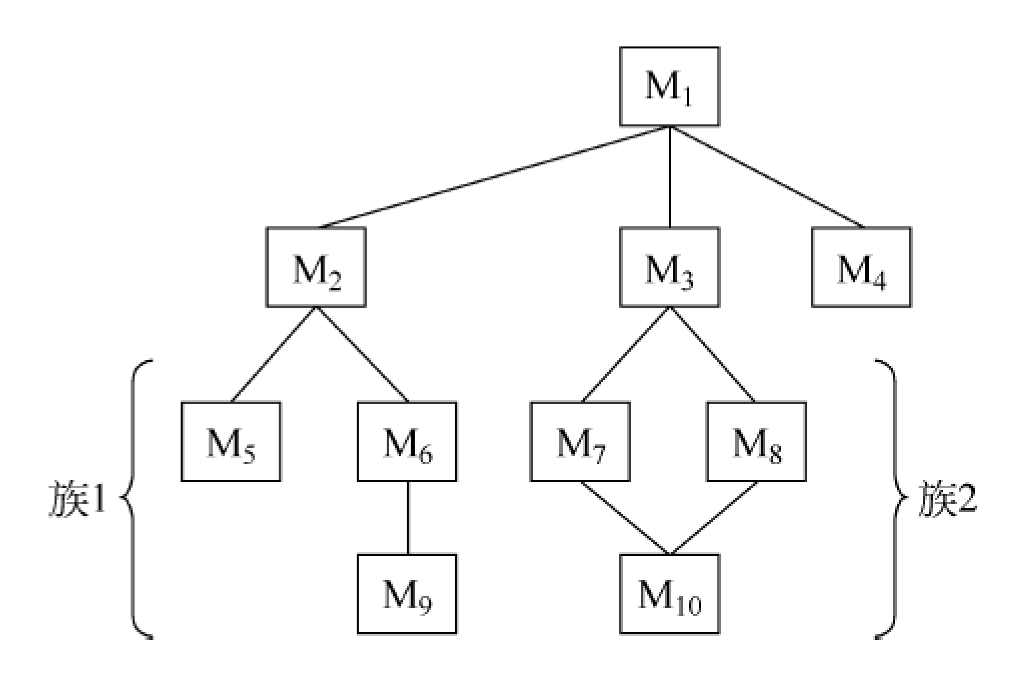


图 4-5 自底向上增量式测试示意图

① 该方法可由下列几个步骤实现。

步骤 1：把底层模块组合成实现某个特定的软件子功能的族。

步骤 2：写一个驱动程序(用于测试的控制程序)，协调测试数据的输入和输出。

步骤 3：对由模块组成的子功能族进行测试。

步骤 4：去掉驱动程序，沿软件结构由下向上移动，把子功能族组合成更大的功能族。

步骤 5：不断重复步骤 2～步骤 4，直到完成。

② 自底向上增量式测试方法的优点如下：

- 虽然模拟中断或异常时还需要设计一定的桩模块，但总体上减少了桩模块的工作量。
- 允许对底层模块行为进行早期验证。
- 在测试初期，可以并行进行集成，从而比使用自顶向下的方式效率高。

③ 自底向上增量式测试方法具有如下缺点。

- 随着逐渐集成到顶层，整个系统变得越来越复杂，对于底层的一些模块将很难覆盖。
- 驱动模块的开发工作量很大。

(3) 混合集成测试

自顶向下增量的方式和自底向上增量的方式各有优缺点。一般来讲，一种方式的优点是另一种方式的缺点，具体测试时通常是把以上两种方式结合起来进行集成和测试，即混合集成测试。

混合集成把系统分为三层，中间一层为目标层。测试时对目标层上面的一层采用自顶向下的集成测试方式，而对目标层下面的一层使用自底向上的集成策略，最后再对目标层进

行测试。

如图 4-6 所示，使用程序桩 S_2、S_3 和 S_4 对用户界面 M_1 进行测试，使用驱动程序 D_5 和 D_6 对底层功能模块 M_7、M_8 和 M_9 进行测试。当整个系统集成时，将程序桩 S_2、S_3 和 S_4 换成中间层模块 M_2、M_3 和 M_4，驱动程序 D_5 和 D_6 换成中间层模块 M_5 和 M_6，从而对中间层的功能模块进行测试。

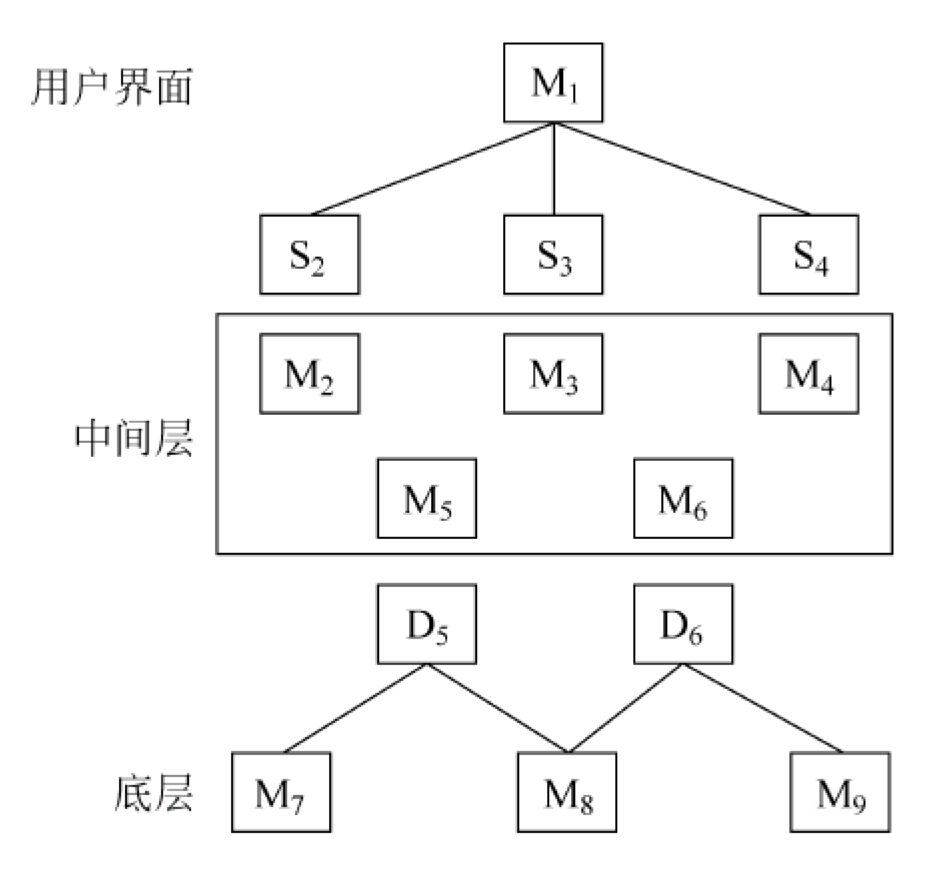


图 4-6 混合集成测试示意图

表 4-2 列出了 3 种集成测试方法的特点对比结果。

表 4-2 3 种集成测试方法的比较

名 称	自顶向下增量式	自底向上增量式	混 合 集 成
集成	早	早	早
基本程序工作时间	早	晚	早
需要驱动程序	否	是	是
需要桩程序	是	否	是
工作并行性	低	中	中
特殊路径测试	难	容易	中
计划与控制	难	容易	难

4.3.3 集成测试方法的对比

1. 非增量式测试与增量式测试

非增量式集成测试模式是先分散测试，然后集中起来一次完成集成测试。如果在模块的接口处存在错误，则会在最后的集成测试时一下子暴露出来。非增量式集成测试时可能发现很多错误，但是为每个错误定位和纠正非常困难，并且在改正一个错误的同时又可能引入新的错误，从而更难断定出错的原因和位置。与此相反，增量式集成测试采用逐步集成和逐步测试的方法，测试的范围逐步增大，从而使错误易于定位和纠正。因此，增量式集成测试比非增量式集成测试有比较明显的优越性。一般不推荐使用非增量式集成测试方法，不

过在规模较小的应用系统中较适用。

2. 自顶向下、自底向下与混合集成测试

自顶向下测试的主要优点在于可以自然地做到逐步求精,从一开始就让测试者了解系统的框架。它的主要缺点是需要提供模拟子模块,被调用模拟子模块可能不能反映真实情况,因此测试有可能不充分。

自底向上测试的优点在于,由于驱动模块模拟了所有调用参数,因此测试数据没有困难。其主要缺点在于,只有到最后一个模块被加入之后才能知道整个系统的框架。

混合集成测试采用自顶向下、自底向上集成相结合的方式,并采取持续集成策略,有助于尽早发现缺陷,提高工作效率。

3. 其他集成测试方法

核心系统先行集成测试能保证一些重要功能和服务的实现,对于快速软件开发有效。采用此种模式的测试,要求系统能明确区分核心软件部件和外围软件部件,从而可以采用高频集成测试,借助于自动化工具实现。

总之,采用自顶向下集成测试和自底向上集成测试方案较为常见。在实际测试工作中,应该结合项目的实际环境及各测试方案适用的范围进行合理的选型。

4.4 确认测试

确认测试的任务是验证软件的有效性,即验证软件功能和性能及其他特点是否与用户的要求一致。在软件需求规格说明书描述了全部用户可见的软件属性,其中有一节叫做有效性准则,它包含的信息就是软件确认测试的基础。

在确认测试阶段需要做的工作如图 4-7 所示。首先要进行有效性测试以及软件配置复审,然后进行验收测试和安装测试,此外,被测软件还要通过专家鉴定,之后才能成为可交付的软件。

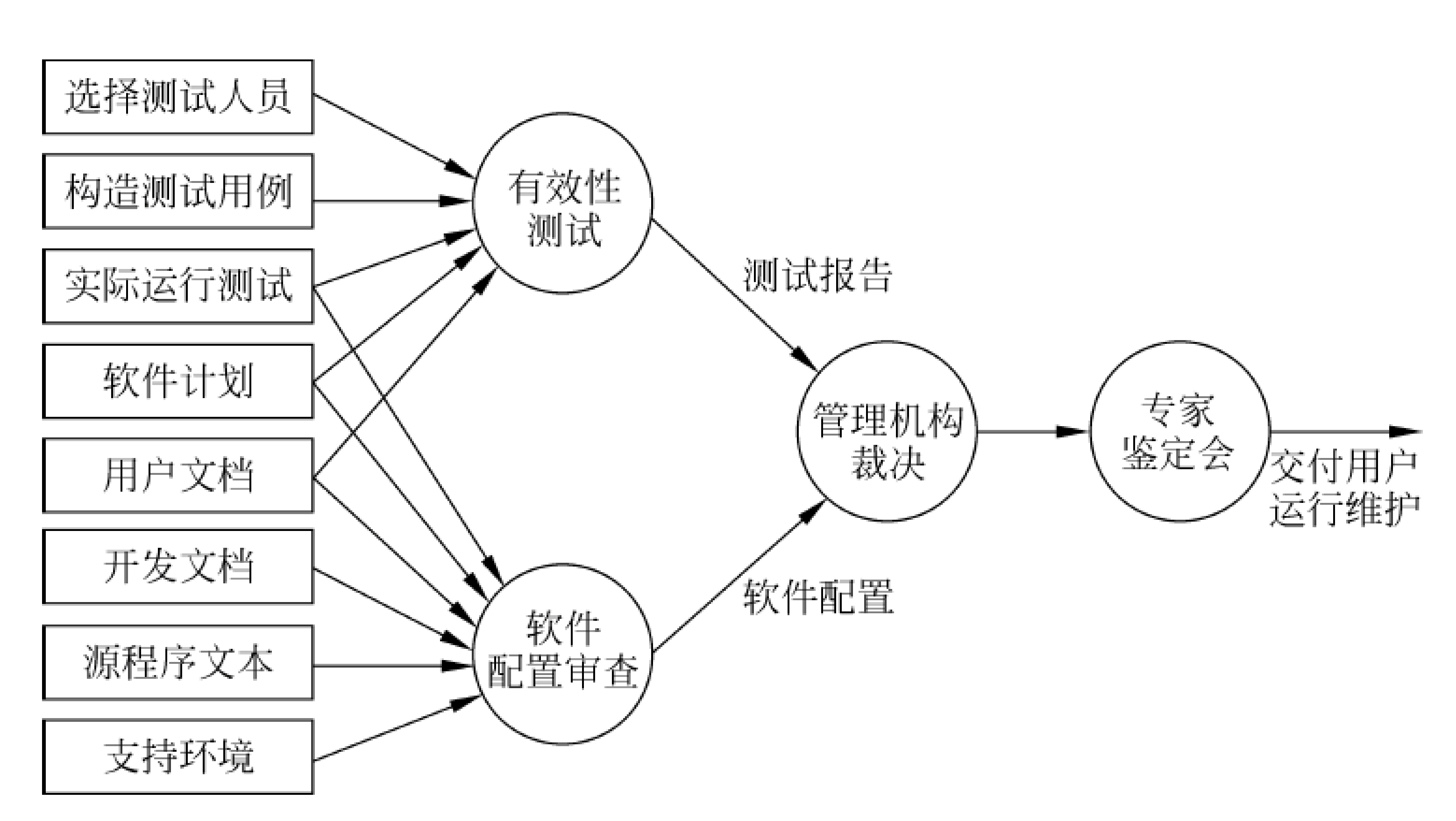


图 4-7 确认测试的步骤

4.4.1 进行有效性测试(功能测试)

有效性测试是在模拟的环境(可能就是开发的环境)下,运用黑盒测试的方法,验证被测软件是否满足需求规格说明书列出的需求。为此,首先需要制定测试计划,规定要做测试的种类。还需要制定一组测试步骤,描述具体的测试用例。通过实施预定的测试计划和测试步骤,确定软件的性能和功能是否与需求相符,并且要求所有的文档都是正确的且便于使用。同时,对其他软件需求,例如可移植性、兼容性、出错自动恢复、可维护性等,也都要进行测试,确认是否满足。

4.4.2 软件配置复查

软件配置复查的目的是保证软件配置的所有成分都齐全,各方面的质量都符合要求,具有维护阶段所必需的细节,而且已经编排好分类的目录。

除了按合同规定的内容和要求,由人工审查软件配置之外,在确认测试的过程中,应当严格遵守用户手册和操作手册中规定的使用步骤,以便检查这些文档资料的完整性和正确性。必须仔细记录发现的遗漏和错误,并且适当地补充和改正。

4.5 系统测试

所谓系统测试,是将通过确认测试的软件,作为整个基于计算机系统的一个元素,与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起,在实际运行(使用)环境下,对计算机系统进行一系列的严格有效的测试,借以发现软件的潜在问题,保证系统的运行。

系统测试明显区别于确认测试。确认测试主要是验证软件功能的实现情况,不考虑各种环境以及非功能问题,如安全性、可靠性、性能等。而系统测试是在更大的范围内进行的测试,着重对系统的性能、特性进行测试。它的目的在于通过与系统的需求定义作比较,发现软件与系统定义不符合或与之矛盾的地方。所以系统测试的测试用例应该根据需求分析规格说明来设计,并在实际使用环境下来运行。

下面对系统测试的内容进行简要介绍。

1. 强度测试

强度测试是要检查在系统运行环境不正常乃至发生故障的情况下,系统可以运行到何种程度的测试。强度测试需要在反常规数据量、频率或资源的方式下运行系统,以检查系统能力的最高实际限度。例如,输入数据速率提高一个数量级,确定输入功能将如何响应;或设计需要占用最大存储量或其他资源的测试用例进行测试。

强度测试的一个变种就是敏感性测试。在程序有效数据界限内的一个小范围内的一组数据可引起极端的或不平稳的错误处理出现,或者导致极度的性能下降的情况发生。敏感性测试即可用来发现可能引起这种不稳定性或不正常处理的某些数据组合。

2. 性能测试

性能测试用来测试软件在系统集成中的运行性能，检查其是否满足需求说明书中规定的性能，特别是对于实时系统或嵌入式系统，仅提供符合功能需求但不符合性能需求的软件是不能接受的。性能测试可以在测试过程的任意阶段进行，即使是在单元层。但只有当整个系统的所有成分都集成在一起后，才能检查一个系统的真正性能。性能测试常常需要与强度测试结合起来进行，并常常要求同时进行硬件和软件检测，这就是说，常常有必要在一种苛刻的资源环境中衡量资源的使用。通常，对软件性能的检测表现在以下几个方面：响应、时间、吞吐量、辅助存储区（例如缓冲区、工作区的大小）、处理精度等。

通过对系统的检测，测试者可以发现导致效率降低和系统故障的原因。外部的测试设备可以检测测试的执行，当出现某种情况时可以记录下来。为了记录性能，需要在系统中安装必要的量测仪表或者为度量性能而设置的软件。

3. 恢复测试

恢复测试是要证实在克服硬件故障（包括断电、硬件或网络出错等）后，系统能否正常地继续进行工作，是否对系统造成任何损害。为此，可采用各种人工干预的手段，模拟硬件故障，故意造成软件出错，并由此检查：系统的错误探测功能——系统能否发现硬件失效与故障；能否切换或启动备用的硬件；在故障发生时能否保护正在运行的作业和系统状态；在系统恢复后能否从最后记录下来的无错误状态开始继续执行作业等。例如断电测试，它的目的是测试软件系统在发生电源中断时能否保护当时的状态且不毁坏数据，然后在电源恢复时从保留的断点处重新进行操作。

4. 安全测试

任何管理敏感信息或者能够对个人造成不正当伤害的计算机系统都是非法侵入的目标。通常力图破坏系统的保护机构以进入系统的主要方法有：正面攻击或从侧面、背面攻击系统中易受损坏的那些部分；以系统输入为突破口，利用输入的容错性进行正面攻击；申请和占用过多的资源压垮系统，以破坏安全措施，从而进入系统；故意使系统出错，利用系统恢复的过程，窃取用户口令及其他有用的信息；通过浏览残留在计算机各种资源中的垃圾（无用信息），获取如口令、安全码、译码关键字等信息；浏览全局数据，从中找到进入系统的关键字；浏览那些逻辑上不存在，但物理上还存在的各种记录和资料等。

安全测试是要检验在系统中已经存在的系统安全性、保密性措施是否发挥作用，有无漏洞，检查系统对非法侵入的防范能力。安全测试期间，测试人员将假扮非法入侵者，采用各种方法试图突破防线。系统安全设计的准则是使非法侵入的代价超过被保护信息的价值。

5. 可靠性测试

软件可靠性是指软件系统在规定的时间内和规定的环境条件下，完成规定功能的能力。它是软件系统的固有特性之一，表明了一个软件系统按照用户的要求和设计目标，执行其功能的可靠程度。软件可靠性与软件缺陷有关，也与系统输入和系统使用有关。从理论上说，

可靠的软件系统应该是正确的、完整的、一致的和健壮的。但是实际上任何软件都不可能达到百分之百的正确，而且也无法精确度量。一般情况下，只能通过对软件系统进行测试来检查其可靠性是否达到预期目标。

可靠性测试是从验证的角度出发，它通过测试验证软件是否达到了用户的可靠性要求，同时发现并纠正影响可靠性的缺陷，以期实现软件可靠性增长。该测试需要从用户的角度出发，模拟用户实际使用系统的情况，设计出系统的可操作视图。

根据在测试过程中收集获得的失效数据，如失效间隔时间、失效修复时间、失效数量、失效级别等，应用可靠性模型，可以得到系统的失效率及可靠性增长趋势。其中可靠性增长趋势是测试开始时的失效率与测试结束时的失效率之比。

从黑盒（占主要地位）和白盒测试两个角度出发有以下几种常用的可靠性模型。

（1）黑盒方面的可靠性模型包括了基本执行时间模型（Musa）、故障分离模型（Jelinski-Moranda）、NHPP 模型及增强的 NHPP 模型（Goel-Okumoto）以及贝叶斯判定模型（Littlewood-Verrall）。

（2）白盒方面的可靠性模型则包括了基于路径的模型和基于状态的模型。

业界流行的可靠性模型还有很多种，不同的可靠性模型其依赖的假设条件和使用范围也不同。对于相同的数据，不同的模型可以得到不同的结果，有些结果可能大相径庭，这往往是因为不同的模型基于的假设条件不同而造成的。对于一个产品，其所适合使用的可靠性模型需要根据实际出发，尽可能选择与可靠性模型假设条件相近的模型。

6. 安装测试

理想情况下，一个软件的安装程序应当平滑地集成用户的新软件到已有的系统中，就像一个客人被介绍到一个聚会中一样，彼此交换适当的问候。虽然有一些对话窗口提供简单的、容易理解的安装选项和支持信息，帮助完成安装过程，然而，在某些糟糕的情况下，安装程序还是可能会出错，新的程序无法工作，已有的功能受到影响，甚至安装过程严重损坏用户系统。

在安装软件系统时，会有多种选择：要分配和装入文件与程序库；布置适用的硬件配置；进行程序的联结。而安装测试就是要找出在这些安装过程中出现的错误，其目的是要验证成功安装系统的能力。它通常是开发人员的最后一个活动，并且通常在开发期间不太受关注。但是，它是客户使用新系统时执行的第一个操作，因此，清晰并且简单的安装过程是系统文档中最重要的部分。

7. 容量测试

容量测试是根据预先分析出反映软件系统应用特征的某项指标极限值（如最大并发用户数、最大数据库记录数等），测试系统在其极限值状态下是否能保持主要功能正常运行。例如，对于编译程序，让它处理特别长的源程序；对于操作系统，让它的作业队列“满员”；对于信息检索系统，让它使用频率达到最大。在使用系统的全部资源达到“满负荷”的情况下，测试系统的承受能力。

容量测试的完成标准可以定义为：所计划的测试已全部执行，而且达到或超出指定的系统限制时没有出现任何软件故障。

8. 文档测试

文档测试旨在检查用户文档(如用户手册)的清晰性和精确性。在用户文档中所使用的例子必须在测试中测试过,确保叙述正确无误。

4.6 验收测试

验收测试是软件产品在完成系统测试后,于发布之前所进行的软件测试活动,它是技术测试的最后一个阶段。通过验收测试后,产品就可进入发布阶段。

验收测试的目的是确保软件准备就绪,可以让最终用户来执行软件的既定功能和任务。它的作用是向未来的用户表明系统能够像预定要求那样工作,应检查软件能否按合同要求进行工作,即是否满足软件需求说明书中的确认标准。

验收测试是以用户为主的测试,由用户参与设计测试用例,在用户界面输入测试数据,并分析测试的输出结果,当然软件开发人员和质量保证人员也应参加。一般使用生产中的实际数据进行测试,在测试过程中,除了考虑软件的功能和性能外,还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

验收测试的结果有两种可能,一种是功能和性能指标满足软件需求说明的要求,用户可以接受;另一种是软件不满足软件需求说明的要求,用户无法接受。项目进行到这个阶段才发现严重错误和偏差一般很难在预定的工期内改正,因此必须与用户协商,寻求一个妥善解决问题的方法,决定必须作很大修改还是在维护后期或下一个版本改进。

验收测试多采用 α 测试和 β 测试,以发现可能只有最终用户才能发现的错误。

4.7 回归测试

软件生命周期中的任何一个阶段,只要软件发生了改变,就可能给该软件带来缺陷问题。这里软件的改变可能是源于发现了错误并做了修改,也有可能是因为在集成或维护阶段加入了新的模块等多种情况。所谓回归测试是一种验证已变更系统的完整性与正确性的测试技术,是指重新执行已经做过的测试的某个子集以保证修改没有引入新的错误或者发现由于更改而引起的之前未发现的错误,也就是保证改变没有带来非预期的副作用。因此,软件开发的各个阶段会进行多次回归测试。

1. 回归测试的实施前提

(1) 当软件中所含错误被发现时,如果错误跟踪与管理系统不够完善,可能会遗漏对这些错误的修改。

(2) 开发者对错误理解得不够透彻,也可能导致所做的修改只修正了错误的外在表现,而没有修复错误本身,从而造成修改失败。

(3) 修改还有可能导致软件未被修改的部分产生新的问题从而产生副作用,使本来工作正常的功能产生错误。

微软公司测试经验表明，一般修复3～4个错误会产生一个新的错误。同样，新代码加入软件的时候，除了新代码有可能含有错误外，还有可能给原有的代码带来影响。因此，软件一旦发生变化，必须重新补充新的测试用例，测试软件功能，确定修改是否达到预期目的，检查修改是否损害原有功能。

2. 回归测试的两个策略

回归测试是贯穿整个测试所有阶段的测试活动，其目的是检验已经发现的缺陷有没有正确修改和修改过程中有没有引发新的缺陷。可以采用如下的策略进行回归测试。

（1）完全重复测试

完全重复测试是指将所有的测试用例全部再完全地执行一遍，以确认问题修改的正确性和修改后周边是否受到影响的测试方法。其缺点是由于要把用例全部执行，因此会增加项目成本，也会影响项目进度，所以很难完全执行。

（2）选择性重复测试

选择性重复测试是指可以选择一部分进行执行，以确认问题修改的正确性和修改后周边是否受到影响的测试方法。下面介绍几种选择性重复测试的方法。

① 覆盖修改法

覆盖修改法是指针对发生错误的模块，选取这个模块的全部用例进行测试。这种方法只能验证本模块是否还存在缺陷，不能保证周边与它有联系的模块不会因为这次改动而引发缺陷。这种回归测试仅根据修改的内容来选择测试用例，仅保证修改的缺陷或新增的功能被实现，其效率最高，但风险也最大，因为它无法保证这个修改是否影响了其他功能。该方法一般用于软件结构设计中模块耦合度较小的情况。

② 周边影响法

周边影响法除了把出错模块的用例执行之外，把周边和它有联系的模块的用例也执行一遍，以保证回归测试的质量，利用该方法需要分析修改可能影响到的那部分代码或功能，对于所有受影响的功能和代码，其对应的所有测试用例都将被回归。至于判断哪些功能或代码受影响，往往依赖于测试人员的经验和开发过程的规范性。

③ 指标达成法

指标达成法是指根据一定的覆盖率指标选择回归测试。例如，规定修改范围内的测试阈值是90%，其他范围内的测试阈值为60%，该方法一般在相关功能影响范围难以界定时使用。

④ 基于操作剖面测试法

如果测试用例是基于软件操作剖面开发的，测试用例的分布情况将反映系统的实际使用情况。回归测试所使用的测试用例个数可以由测试预算确定，可以优先选择针对最重要或最频繁使用功能的那些测试用例，尽早发现对可靠性有最大影响的故障。

⑤ 基于风险选择测试法

该方法根据缺陷的严重性来进行测试，基于一定的风险标准从测试用例库中选择回归测试包。选择最重要、关键以及可疑的测试，而跳过那些次要的、例外的测试用例或功能相对稳定的模块。

3. 回归测试的流程

回归测试的流程一般具有如下步骤。

步骤1：在测试策略制定阶段，制定回归测试策略。

步骤2：确定回归测试版本。

步骤3：回归测试版本发布，按照回归测试策略执行回归测试。

步骤4：回归测试通过，关闭缺陷跟踪单。

步骤5：回归测试不通过，缺陷单返回；开发人员重新修改，再次做回归测试。

每当一个新的模块被当作集成测试的一部分加进来的时候，软件就发生了改变。新的数据流路径建立起来，新的I/O操作可能也会出现，还有可能激活新的控制逻辑。这些改变可能会使原本工作得很正常的功能产生错误。在集成测试策略的环境中，回归测试是对某些已经进行过的测试的某些子集再重新进行一遍，以保证改变不会传播无法预料的副作用。

4. 回归测试与一般测试的比较

回归测试与一般测试相比两者具有许多不同点。

(1) 测试计划的可获性：一般测试都会有系统规格说明书和测试计划，通常一般测试的测试用例都是新的；而对于回归测试，可能面临的是更改了的规格说明书、修改过的程序和需要更新的测试计划。

(2) 范围：一般测试的目标是检测整个程序的正确性；而回归测试的目标是检测被修改的相关部分的正确性以及它与系统原有功能的整合。

(3) 时间分配：一般测试所需时间通常是在软件开发之前预算好的；而回归测试所需的时间(尤其是修正性的回归测试)往往不包含在整个产品进度表中。

(4) 开发信息：一般测试关于开发的知识和信息可随时获取；而回归测试由于可能会在不同的地点和时间进行，需要保留开发信息以保证回归测试的正确。

(5) 完成时间：由于回归测试只需测试程序的一部分，因此完成测试时间通常比一般测试所需时间少。

(6) 执行频率：回归测试在一个系统的生命周期内往往要多次进行，一旦系统经过修改就需要进行回归测试。

4.8　本章小结

软件测试贯串软件产品开发的整个生命周期，软件项目一开始软件测试也就开始了。从过程来看，软件测试是由一系列的不同测试阶段所组成的。包括制定测试计划、测试设计、单元测试、集成测试、确认测试、系统测试、验收测试和回归测试等。软件开发的过程是自顶向下的，测试则正好相反，其过程是自底向上、逐步集成的。

单元测试是软件测试过程中最早期的测试活动，此阶段集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能。

集成测试是将已分别通过测试的单元按设计要求组合起来再进行测试，以检查这些单

元之间的接口、参数传递是否存在问题。

确认测试是检查已实现的软件是否满足了需求规格说明中确定了的各种需求和功能，以及软件配置是否完全、正确。

系统测试就是充分运行或模拟运行软件系统，以验证系统是否满足产品的质量需求，特别是非功能性的质量需求。

验收测试是软件产品在完成了功能测试和系统测试之后，发布之前所进行的软件测试活动。验收测试的重要特征就是用户参与。

回归测试是由于软件修改或变更而对修改后的工作版本中所有可能影响的范围进行的测试。回归测试伴随测试全程，一旦变更或修改，都要进行相应的回归测试。

习题 4

1. 软件测试的步骤是什么？各阶段关系如何？
2. 单元测试有哪些内容？测试中采用什么方法？
3. 集成测试方法有几种？集成测试与单元测试的区别是什么？
4. 采用自顶向下的增量方式将模块按系统程序结构进行组装，其步骤是什么？
5. 什么是回归测试？什么时候进行回归测试？回归测试与一般测试的区别是什么？

第5章 测试用例设计

Grenford J. Myers 在《The Art of Software Testing》一书中提到：一个好的测试用例是指很可能找到迄今为止尚未发现的错误的测试用例。由此可见测试用例设计工作在整个测试过程中的地位，我们不能只凭借一些主观或直观的想法来设计测试用例，应该要以一些比较成熟的测试用例设计方法为指导，再加上设计人员个人的经验积累，二者相结合才能设计出优秀的测试用例。

5.1 测试用例的基本概念

测试用例（Test Case）是为特定的目的而设计的一组测试输入、执行条件和预期的结果的程序代码，测试用例是执行测试的最小实体。简单地说，测试用例就是设计一个场景，使软件程序在这种场景下，必须能够正常运行并且达到程序所设计的执行结果。

测试用例目前没有经典的定义，比较通常的说法是：对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略。内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。

要使最终用户对软件感到满意，最有力的举措就是对最终用户的期望加以明确阐述，以便对这些期望进行核实并确认其有效性，测试用例反映了要核实的需求。核实这些需求可能通过不同的方式并由不同的测试员来实施。例如，执行软件以便验证它的功能和性能，这项操作可能由某个测试员采用自动测试技术来实现；计算机系统的关机步骤可通过手工测试和观察来完成；不过，市场占有率和销售数据（以及产品需求），只能通过评测产品和竞争对手的销售数据来获取。

既然可能无法（或不必负责）核实所有的需求，那么为测试挑选最适合或最关键的需求关系到项目的成败。选中要核实的需求将是对成本、风险和对该需求进行核实的必要性这三者权衡考虑的结果。根据测试过程中具体设计到问题类型及测试需求，可将测试用例分为如下几类：

（1）功能性测试用例。

（2）界面测试用例，适用于所有测试阶段中的界面测试。

（3）数据处理测试用例，适用于所有测试阶段中的数据处理测试。

（4）操作流程测试用例，适用于所有流程性测试。

（5）安装测试用例，适用于所有安装测试。

测试用例构成了设计和制定测试过程的基础。测试的“深度”与测试用例的数量成比例，由于每个测试用例反映不同的场景、条件或经由产品的事件流，因而，随着测试用例数量的增加，您对产品质量和测试流程也就越有信心。判断测试是否完全的一个主要评测方法是考察对需求的覆盖，而这又是以确定、实施或执行的测试用例的数量为依据的。类似下面这样的说明：“95%的关键测试用例已得到执行和验证”，远比“我们已完成95%的测试”更有意义。

5.2 测试用例的设计

5.2.1 测试用例设计说明

测试工作量与测试用例的数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排。测试用例通常根据它们所关联的测试类型或测试需求来分类，而且将随类型和需求进行相应的改变。最佳方案是为每个测试需求至少编制两个测试用例：一个测试用例用于证明该需求已经满足，通常称作正面测试用例；另一个测试用例反映某个无法接受、反常或意外的条件或数据，用于论证只有在所需条件下才能够满足该需求，这个测试用例称作负面测试用例。

测试用例是软件测试的核心，不同类别的软件测试的用例是不同的。一个好的测试用例应具有以下优点：

(1) 在开始实施测试之前设计好测试用例，避免盲目的测试。

(2) 测试用例应使软件测试的实施重点突出、目的明确。

(3) 可根据测试用例的多少和执行难度，估算测试工作量，便于管理与跟踪测试项目的时间和资源。

(4) 减少回归测试的复杂程度。

(5) 在软件版本更新后只需修正少量的测试用例便可展开测试工作、降低工作强度、缩短项目周期。

(6) 功能模块测试用例的通用化和复用化会使软件测试易于开展。

(7) 根据测试用例的操作步骤和执行结果，可以方便地书写软件测试缺陷报告。

(8) 可以根据测试用例的执行等级，实施不同级别的测试。

(9) 可为分析软件缺陷和程序模块质量提供依据。

(10) 可以最大限度地找出软件隐藏的缺陷。

(11) 测试用例内容清晰、格式一致、分类组织。

正确地认识测试十分重要，如果只是为了表明程序是正确的而以此为出发点进行测试，就会设计一些不易暴露错误的测试方案；相反，如果测试是为了发现程序中的错误，就会力求设计出最能暴露错误的测试方案。

测试的目的决定了测试方案的设计，G. Myers给出了关于软件测试目的的观点：

(1) 测试是为了发现程序中的错误而执行程序的过程。

(2) 好的测试方案是极有可能且尽可能多地发现迄今尚未发现的错误的测试。

(3) 成功的测试是发现了迄今尚未发现的错误的测试。

怎样才能实现测试目的呢？为了设计出有效的测试方案，软件工程师必须深入理解并正确运用指导软件测试的基本准则。这些基本原则如下：

(1) 测试用例应具有代表性：即测试用例能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的，以及极限的输入数据、操作和环境设置等。

(2) 测试结果具有可判定性：即测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。

(3) 测试结果具有可再现性：即对同样的测试用例，系统的执行结果应当是相同的。

5.2.2 测试用例的编写标准

在制定测试计划时，由于不同软件公司的背景不同，测试计划内容会有差异，但一些基本内容是相同的。在 ANSI/IEEE 列出了测试用例编写规范和模版，模版中主要元素有：

1. 标识符

每个测试用例应该有一个唯一的标识符，它将成为所有和测试用例相关的文档、表格引用与参考的基本元素，这些文档包括缺陷报告、测试任务、测试报告等。

2. 测试项(Test Item)

测试用例应该准确地描述所需要测试的项及其特征，测试项应该比测试设计说明中所列出的特征描述更加具体。

3. 测试环境要求

用来表征执行该测试用例需要的测试环境，一般来说，在整个测试模块里面应该包含对整个测试环境的特殊需求，而单个测试用例的测试环境需要表征该测试用例单独所需要的特殊环境需求。

4. 输入标准

用来执行测试用例的输入要求。这些输入可能包括数据、文件或者操作。

5. 输出标准

标识按照指定的环境、条件和输入而得到的期望输出结果。如果可能的话，尽量提供适当的系统规格说明来证明期望的结果。

6. 测试用例之间的关联

用来标识该测试用例与其他测试用例之间的依赖关系。在测试的实际过程中，很多的测试用例并不是单独存在的，它们之间可能有某种依赖关系。例如测试用例 A 需要在测试用例 B 的测试结果正确的前提下才能被执行，此时测试人员需要表明测试用例 A 对测试用例 B 的依赖，从而保证测试用例的严谨性。

5.2.3 测试用例设计考虑的因素

在测试用例的组织编写过程中，尽量考虑有代表性的典型测试用例，以求实现以点带面的效果，这就要求在测试用例设计过程中考虑一些基本因素。

（1）测试用例必须具有代表性、典型性。一个测试用例应能基本涵盖一组特定的情形，目标明确，这可能要借助有效的测试用例设计方法和对用户使用产品的准确把握。

（2）寻求系统设计、功能设计的弱点。测试用例需要确切地反映功能设计中可能存在的各种问题，而不是简单复制产品规格设计说明书的内容。同时，测试用例还需要按照功能规格说明书的要求进行设计，将所有可能的情况结合起来考虑。

（3）测试用例需要考虑正确的输入，也要考虑到错误的输入或异常的输入，需要分析怎样能够使这样的错误或者异常发生。

（4）对于用户测试用例，要多考虑用户实际使用场景。用户测试用例是基于用户的实际可能场景，从用户的角度来模拟程序的输入，从而针对程序进行的测试用例。用户测试用例不仅需要考虑用户实际的环境因素（例如在 Web 程序中需要对用户的连接速度、负载进行模拟），还需要考虑各种网络连接方式的速度。在执行本地化测试时，需要充分考虑用户所在国家、地区的风俗、语言以及习惯用法。

5.2.4 测试用例设计的基本原则

除了需要遵守基本的测试用例编写规范外，在测试用例设计时，还需要遵循一些基本的原则。

1. 尽量避免含糊的测试用例

含糊的测试用例给测试过程带来困难，甚至会影响测试的结果。在测试过程中，测试用例的状态是唯一的，通常情况下，良好的测试用例一般会有三种状态：通过（Pass）、未通过（Failed）以及未进行测试（Not Done）。如果测试未通过，一般会有测试的错误报告进行关联；如果未进行测试，则需要说明原因。总之，清晰的测试用例使测试人员在测试过程中不会出现模棱两可的情况，不能说某个测试用例部分通过，部分没有通过，或者软件错误出现在这个测试用例，但测试用例描述中却不能找出问题。这样的测试用例将给测试人员的判断带来困难，同时也不利于测试过程的跟踪。

2. 尽量将具有相类似功能的测试用例抽象并归类

在前面我们一直强调软件测试过程是无法进行穷举测试的，因此，学会对相类似的测试用例进行抽象并归类显得尤为重要，一个好的测试用例应该能代表一组或一系列的测试过程。

3. 尽量避免冗长和复杂的测试用例

当测试用例包含很多不同类型的输入或输出时，测试过程的逻辑变得复杂而不连续，此时，需要对测试用例进行分解，保证验证结果的唯一性，便于跟踪和管理。

在实际的测试用例设计中，需要将前面的基本原则和考虑因素结合起来，按照实际测试需求灵活地组织设计测试用例。

5.2.5　测试用例的分类

测试用例可分为白盒测试用例和黑盒测试用例。黑盒测试用例又可分为功能测试用例和非功能测试用例。功能测试用例主要有等价类划分、边界值分析、因果图法、判定表法、场景法、正交实验法、随机测试法和错误推测法等。非功能测试用例主要有配置/安装测试、兼容性测试、互操作性测试、文档和帮助测试、性能测试、可靠性测试、易用性测试和界面测试等。

1. 白盒测试用例

白盒测试用例主要有由逻辑覆盖法和基本路径测试法设计的测试用例，设计的基本思路是使用程序设计的控制结构导出测试用例。

2. 测试软件各项功能的测试用例

例如，文字编辑器中的新建文档功能、打开文档功能、保存文档功能、打印功能、编辑功能等。功能测试用例一般采用等价类划分法、边界值分析法、错误推测法、因果图法等进行设计，这些都属于黑盒测试技术。

3. 用户界面测试用例

大部分客户对界面的要求非常高，所以对于测试人员来说，也必须特别注意界面的美观问题。在测试过程中，应注意以下几点：

- 界面的线条是否一致，每个界面中的线条是否对齐；
- 整个系统的界面是否保持一致；
- 界面中是否存在错别字；
- 界面所有的按钮样式是否一致；
- 操作是否友好；
- 界面所有的按钮、下拉框是否都能响应；
- 界面所有的链接是否正常；
- 界面所有的输入框是否都能进行校验(例如搜索框、字段输入框)；
- 界面所有的列表页标题字是否会折行；
- 界面所有展示的图片是否样式一致；
- 浏览器的兼容性问题，检查页面在不同浏览器下是否会发生异常；
- 每个页面的提示字体的颜色、格式是否统一准确；
- 确保用户界面符合公司和行业的标准。

4. 软件的各项非功能测试用例

软件的非功能属性不描述软件的功能，它是站在整体的角度说明软件应满足的要求，除了上面提到的用户界面测试之外，它还包括：性能测试、兼容性测试、安装测试、安全性测

试、文档测试。

5.3 测试用例设计实例

对于测试人员来说，测试用例的设计编写是一项必须掌握的能力。但有效地设计和熟练地编写测试用例是一项十分复杂的技术，测试用例编写者不仅要掌握软件测试的技术和流程，而且还要对整个软件的业务，以及对被测软件的设计、功能规格说明、用户使用场景及程序模块结构等方面，都有比较透彻的理解和明晰的把握，稍有不慎就会顾此失彼，造成疏漏。

测试用例的设计方法不是单独存在的，具体到每个测试项目都会用到多种方法，如同每种类型的软件有其各自的特点，每种测试用例设计方法也有各自的特点，于是，针对不同软件就有不同的测试用例设计方法。测试用例的编写如表 5-1 所示。

表 5-1　测试用例编写表

测试标题			用例的编号 ID		
测试技术		测试环境		特殊要求	
测试用例设计人员		测试人员		测试日期	
测试目的					
测试对象					
测试项	测试内容	测试方法与步骤		测试判断准则	测试结果

【例 5-1】 求找零钱最佳组合：假设商店货品价格(R)皆为不大于 100(单元为元)的整数，若顾客付款在 100 元内(P)，求找给顾客的最少货币张数？试根据边界值法设计测试用例。(注，货币面值有四种：50 元为 N_{50}，10 元为 N_{10}，5 元为 N_5，1 元为 N_1)。

解：(1) 分析输入的边界情况：

$R>100$　　$0<R\leqslant 100$　　$R\leqslant 0$

$P>100$　　$R\leqslant P\leqslant 100$　　$P<R$

(2) 分析零钱最佳组合的输出情况：

$N_{50}=1$　　$N_{50}=0$

$1\leqslant N_{10}\leqslant 4$　　$N_{10}=0$

$N_5=1$　　$N_5=0$

$1\leqslant N_1\leqslant 4$　　$N_1=0$

(3) 分析找出每一个决策点，以 RR_1、RR_2、RR_3 分别表示要找 50、10、5 元货币时的剩余找零金额。

$R>100$　　$R\leqslant 0$

$P>100$　　$P<R$

$RR_1\geqslant 50$　　$RR_2\geqslant 10$　　$RR_3\geqslant 5$

（4）根据上述的输入/输出条件组合出可能的情况：

$R>100$

$R\leqslant 0$

$0<R\leqslant 100$　　$R\leqslant P\leqslant 100$　　$RR=50$

$0<R\leqslant 100$　　$R\leqslant P\leqslant 100$　　$RR=49$

$0<R\leqslant 100$　　$R\leqslant P\leqslant 100$　　$RR=10$

$0<R\leqslant 100$　　$R\leqslant P\leqslant 100$　　$RR=9$

$0<R\leqslant 100$　　$R\leqslant P\leqslant 100$　　$RR=5$

$0<R\leqslant 100$　　$R\leqslant P\leqslant 100$　　$RR=4$

$0<R\leqslant 100$　　$R\leqslant P\leqslant 100$　　$RR=1$

$0<R\leqslant 100$　　$R\leqslant P\leqslant 100$　　$RR=0$

（5）为满足以上各种情形，设计测试用例如表 5-2 所示。

表 5-2　设计测试用例 1

测 试 用 例	货品价格 R	付款金额 P
Test1	101	—
Test2	0	—
Test3	−1	—
Test4	100	101
Test5	100	99
Test6	50	100
Test7	51	100
Test8	90	100
Test9	91	100
Test10	95	100
Test11	96	100
Test12	99	100
Test13	100	100

【例 5-2】 针对下面 Test 函数按照基本路径测试方法设计测试用例。

```
int Test(int i_count, int i_flag)
{
    int i_temp = 0;
    while(i_count > 0)
    {
        if(i_flag == 0)
        {
            i_temp = i_count + 100;
            break;
        }
```

```
        else
        {
            if(i_flag == 1)
            {
                i_temp = i_temp + 10;
            }
            else
            {
                i_temp = i_temp + 20;
            }
        }
        i_count -- ;
    }
    return i_temp;
}
```

解：首先标记出路径节点。

```
int Test(int i_count, int i_flag)
{
1   int i_temp = 0;
2   while(i_count > 0)
    {
3       if(i_flag == 0)
        {
4           i_temp = i_count + 100;
5           break;
        }
6       else
        {
7           if(i_flag == 1)
            {
8               i_temp = i_temp + 10;
            }
9           else
            {
10              i_temp = i_temp + 20;
            }
        }
11      i_count -- ;
    }
12  return i_temp;
}
```

得到程序控制流程图如图 5-1 所示。

程序环路复杂度：CC＝4

基本路径集：Path1 1-2-3-6-7-8-11-2-12

Path2 1-2-12

Path3 1-2-3-4-5-12

Path4 1-2-3-6-7-9-10-11-2-12

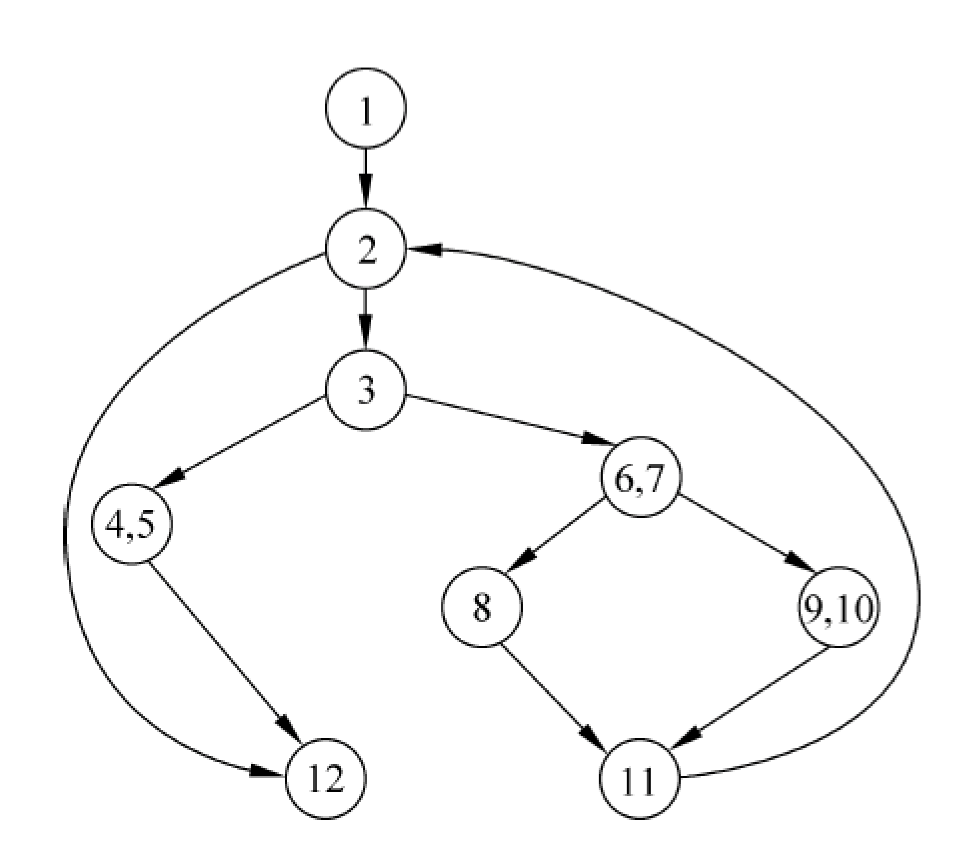


图 5-1 Test 程序控制流程图

因此设计测试用例如表 5-3 所示。

表 5-3 设计测试用例 2

用例 ID	i_count	i_flag	预 期 输 出
Test1	1	1	10
Test2	0	2	0
Test3	2	0	102
Test4	1	3	20

【例 5-3】 如图 5-2 所示,有 4 条可执行语句,设计测试用例要求完成条件组合覆盖。

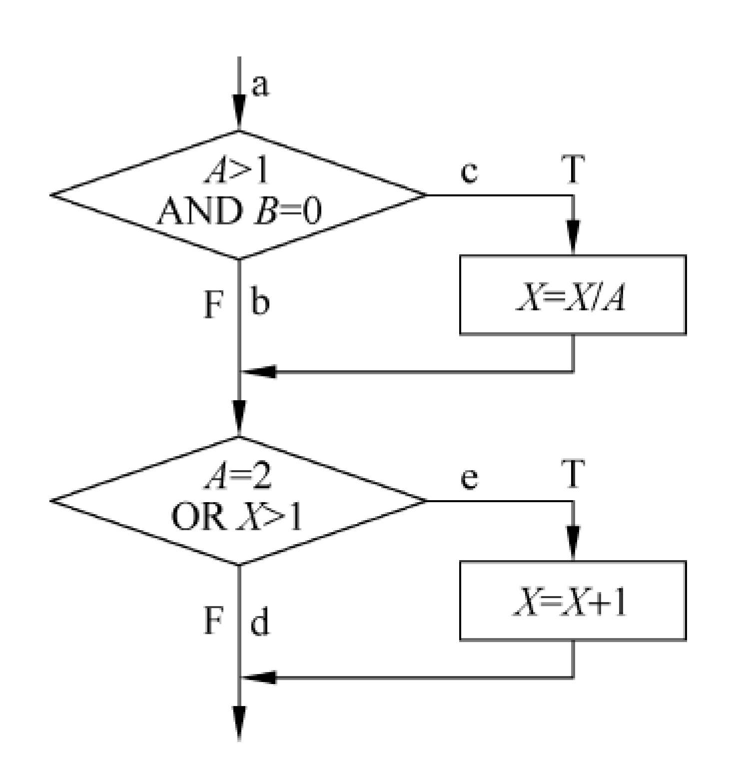


图 5-2 条件组合覆盖例题

解:通过分析得到,第一个判定($A>1$ AND $B=0$)包含了两个条件,测试中应考虑各种条件取值的情况:

$A>1$ 为真,记为 M_1。

$A>1$ 为假,记为 $-M_1$。

$B=0$ 为真,记为 M_2。

$B=0$ 为假,记为 $-M_2$。

对于第二个判定,($A=2$ OR $X>1$)也应考虑如下情况:

$A=2$ 为真，记为 M_3。

$A=2$ 为假，记为 $-M_3$。

$X>1$ 为真，记为 M_4。

$X>1$ 为假，记为 $-M_4$。

将图 5-2 给出的多重条件判定分解，形成图 5-3 所示的由多个基本判定组成的流程图，这样就可以有效地检查所有的条件是否正确，实现条件组合覆盖。设计的测试用例见表 5-4。

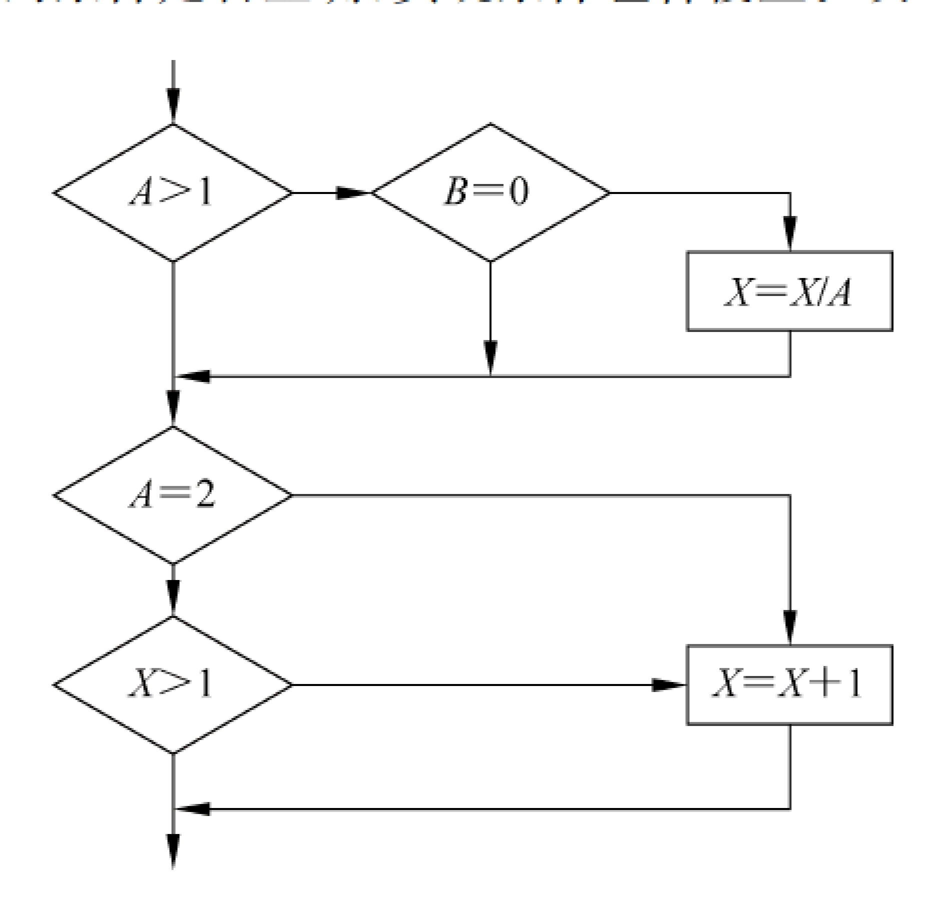


图 5-3 分解为由基本判定组成的流程图

表 5-4 设计测试用例 3

A,B,X	执行路径	覆盖条件
2,0,3	ace	M_1,M_2,M_3,M_4
2,1,1	abe	$M_1,-M_2,M_3,-M_4$
1,1,1	abd	$-M_1,-M_2,-M_3,-M_4$
1,0,3	abe	$-M_1,M_2,-M_3,M_4$

【例 5-4】 有一个 B/S 结构的登录界面，当用户在地址栏输入相应地址，显示登录界面后，要求输入用户名和密码登录，系统会对输入内容自动校验，并给出提示信息；如果用户名或密码任一信息未输入，也给出相应提示信息；如果连续 3 次未通过验证，则自动关闭 IE。根据以上情况，设计登录界面测试用例。

通过分析可知该案例是以一个 B/S 结构的登录功能点为被测对象，所需测试为黑盒测试，假设用户使用的浏览器为 IE 6.0，设计出登录界面测试用例，如表 5-5、表 5-6 所示。

表 5-5 登录界面测试用例

用例 ID	XXXXXX	用例名称	系统登录
用例描述	系统登录		
	在用户名存在、密码正确的情况下，进入系统		
	页面信息包含：页面背景显示		
	用户名和密码录入接口，输入数据后登录系统		
用例入口	打开 IE，在地址栏输入相应地址		
	进入该系统登录页面		

表 5-6　具体测试项

用例 ID	场　　景	测 试 步 骤	预 期 结 果	备注
TC1	初始页面显示	从用例入口处进入	显示与详细设计一致	
TC2	用户名登录验证	输入已存在的用户：zhangsan	输入成功	
TC3	用户名容错性验证	输入：xxadgddddd dddddededdddd	输入到一定长度时，拒绝输入	超过规定长度
TC4	密码录入	输入与用户名相关联的密码：123	输入成功	
TC5	系统登录成功	TC2，TC4，单击登录按钮	登录系统成功	
TC6	用户名、密码校验	输入不存在的用户名、密码登录	登录失败	提示信息 1
TC7	登录密码校验	输入用户名但未输入密码	登录失败	提示信息 2
TC8	密码有效性校验	输入用户名和密码不一致	登录失败	提示信息 3
TC9	用户有效性校验	输入不存在的用户名、密码	登录失败	提示信息 4
TC10	系统登录安全校验	连续 3 次未成功	登录失败	提示信息 5

5.4　测试用例的执行与跟踪

测试用例最终是为实现有效的测试服务的，那么怎样将这些测试用例完整地结合到测试过程中加以使用呢？这就是测试用例的执行、跟踪和维护问题。

搭建测试环境之后，根据定义的顺序，即可逐个执行测试用例。测试用例执行中应该注意以下几个问题。

- 全方位地观察测试用例执行结果；
- 加强测试过程记录；
- 及时确认发现的问题；
- 与开发人员良好沟通；
- 及时更新测试用例；
- 提交一份优秀的问题报告单；
- 测试结果分析。

完成测试实施后，需要对测试结果进行评估，并且编制测试报告。

5.4.1　执行测试用例

图 5-4 展示了测试用例的执行过程。

最终形成的测试结果就是所需要跟踪和分析的测试输出。

5.4.2　跟踪测试用例

在计划确定后开始执行测试之前，测试组长应该能够回答下面几个问题：

- 测试中需要执行哪些测试组件？
- 测试计划中有多少测试用例？

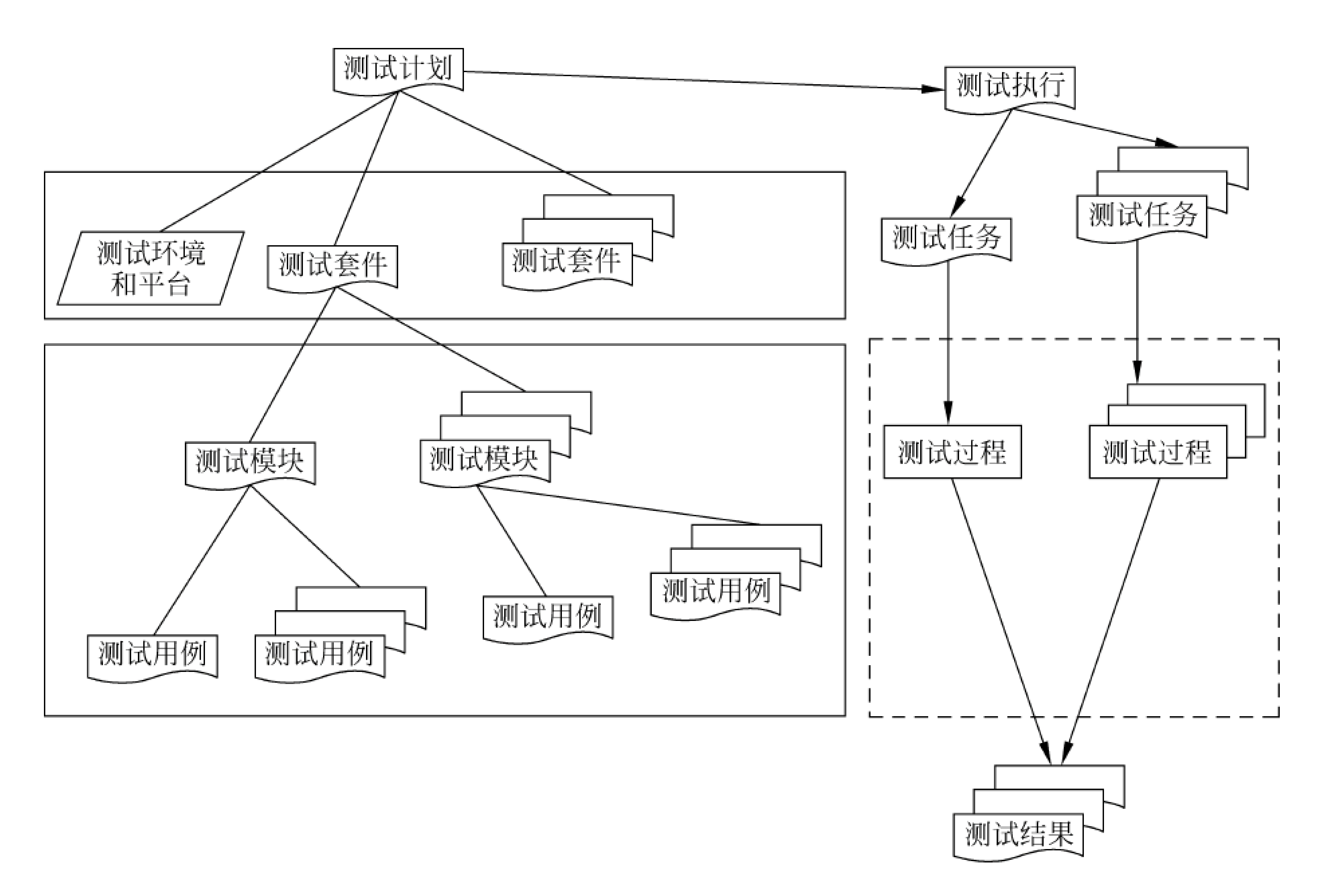


图 5-4 测试用例的执行过程

- 在执行测试过程中,使用什么方法来记录测试用例的状态?
- 如何挑选出有效的测试组件和测试用例来着重测试某些模块?
- 上一次使用的测试用例的通过率是多少?
- 在未通过的测试用例中,有多少是上一次执行的时候也未通过的?

测试用例的跟踪主要是针对测试过程中测试用例的执行和输出而进行的,它使得整个测试过程被有效管理,并可实现测试有效性评估。跟踪测试用例包括以下两方面的内容。

1)测试用例执行的跟踪

在测试用例执行过程中,实现测试用例执行过程的跟踪可以有效地将测试过程量化。例如,执行一轮测试中,需要跟踪总共执行了多少测试用例,每个测试人员平均每天使用多少测试用例,测试用例通过/未通过/未使用的占多少,未使用的原因是什么,当然,这是个相对的过程,测试人员工作量的跟踪不应该仅仅凭借测试用例的执行情况和发现的程序缺陷多少来判定,但至少通过测试执行情况的跟踪可以大致判定当前的项目的质量与进度,并对测试时间做出大致的推断。

2)测试用例覆盖率的跟踪

测试用例的覆盖率指的是根据测试用例进行测试的执行结果与实际的软件存在的问题的比较,通过其可实现对测试有效性的评估。在一个测试的执行中,测试用例通过率为92%,测试用例未通过率为5%,测试用例未使用率3%,在发现的软件缺陷和错误中,有90%通过测试用例检测出来,10%是未通过测试用例检验出来,此时,需要对这些软件错误进行分类和数据分析,完善测试用例,从而使得测试结果更准确,让遗漏问题的可能性最小化。

根据实际数据分析,可以对两个模块进行单独测试,通过纵向的数据比较,实现软件质

量的管理和追踪。跟踪测试用例的形式一般有以下几种。

（1）记忆。

即凭借个人的记忆来跟踪测试用例，这是一种不太可取的办法，除非测试只是针对个人开发的小型软件。

（2）书面文档。

在比较小规模的测试项目中，使用书面文档记录和跟踪测试用例也是一种可行的方法。测试用例清单的列表和图例也可以被有效地使用，但作为组织和搜索数据进行分析时，这种方法是很有局限的。

（3）电子表格。

一种流行而高效的方法是使用电子表格来跟踪和记录测试的过程，通过表格中列出的测试用例的跟踪细节，可以直观地看到测试的状态以及分析和统计测试用例的通过、与软件缺陷的关联等，这为测试中有效管理和分析测试过程以及软件的质量提供了有效的量化依据。

（4）自定义数据库。

最理想的方式是通过自定义的数据库来跟踪测试用例的执行和覆盖率。例如，测试人员通过特定的自定义程序将测试的结果提交，通过自定义的数据库来存储这些测试结果，并通过自己编写的工具生成报表、分析图等，这样将更加有效地管理和跟踪整个测试过程，当然，所花费的成本也是最高的。

5.4.3　维护测试用例

良好的测试用例一般具有很强的可重用性，但是在重复使用的过程中，需要对测试用例进行维护或者更新。测试用例不是一成不变的，当一个阶段测试过程结束后，或多或少会发现一些测试用例编写得不够合理，或者当在下一个版本中使用前一个版本的测试用例，由于部分功能发生了改变，也需要修改测试用例，使之具有良好的延续性。通常情况下，测试用例需要更新可能有以下几种原因：

（1）先前的测试用例设计不全面或不够准确。

（2）部分很严重的软件错误未在测试用例中涵盖。

（3）新版本有新的功能需求或者改动。

（4）编写的测试用例不规范。

（5）旧的测试用例已经不再适用，需要删除。

维护测试用例的过程是实时的、长期的，和编写测试用例不同，维护测试用例一般不涉及到大的组织结构的改动。例如在某个模块里，如果先前的测试用例不能覆盖目前的测试内容，可能需要新定义一个独立的测试模块单元来重新组织新的测试用例。和测试用例编写过程相同的是，测试用例的维护需要以下几个步骤。

（1）发现测试用例中错误或不合理的地方，向编写者提出测试用例修改建议，并提供充分的理由。

（2）测试用例编写者根据测试用例的关联性和修改意见，进行测试用例的修改。

（3）向开发项目组长递交修改后的测试用例。

（4）项目组长、开发人员以及测试用例编写者进行复核后提出意见，通过后，由测试用

例编写者进行最后的修改，并提供修改后的文档和修改日志。

上述测试用例维护的过程可总结为图 5-5。

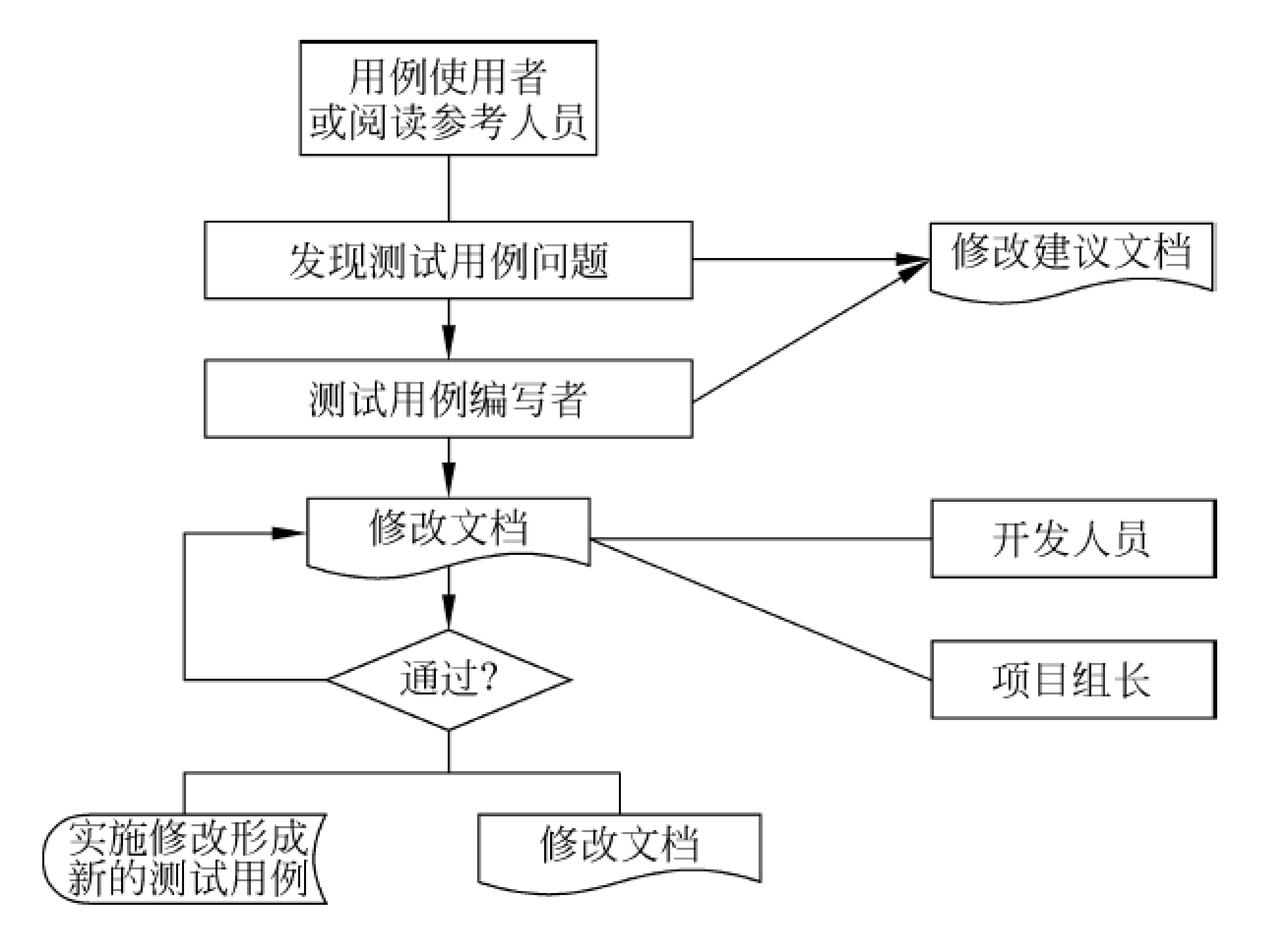


图 5-5　测试用例维护流程图

5.5 测试用例管理

测试用例管理主要包括以下几个方面：

- 测试用例的组织方式；
- 测试用例的评审；
- 测试用例的修改更新；
- 测试用例的管理软件。

测试用例管理示意图如图 5-6 所示。

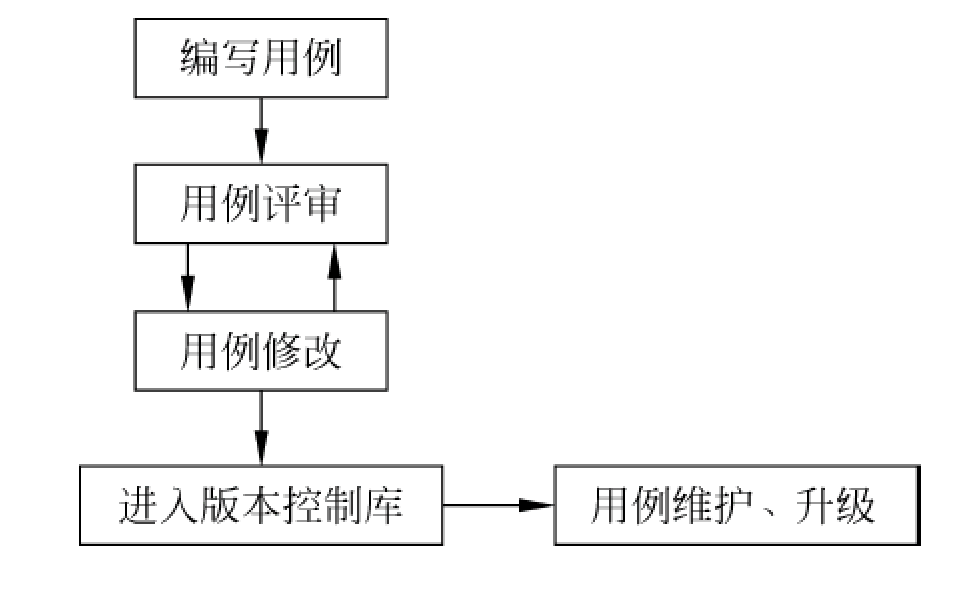


图 5-6　测试用例管理示意图

1. 测试用例的组织

进行测试用例的组织可使用自顶向下的设计方法，首先由测试计划实现测试设计说明书，再通过具体的测试设计说明书实现测试用例的规格说明书，由规格说明书来编写具体的测试用例，如图 5-7 所示。

测试用例必须有效地组织起来，才能发挥效率。通常情况下，使用以下几种方法来组织测试用例：

(1) 按照程序的功能模块组织：应用程序的规格说明书一般是按照不同的功能块进行组织的，因此，按照程序的功能块进行测试用例的组织是一种很好的方法。将属于不同模块的测试用例组织在一起，能够很好地覆盖所测试的内容，准确地执行测试计划。

(2) 按照测试用例的类型组织：在测试过程中可以将所有功能/逻辑测试、压力/负载测试、异常测试、兼容性测试等的相同类型的用例组织成单独的测试单元或模块来测试。

(3) 按照测试用例的优先级组织：测试用例具有不同的优先级，可以按照实际测试过程中的需要，自己定义测试用例的优先级，从而使得测试过程有层次、有主次地进行。

以上三种方法中，根据程序模块进行组织是最常用的方法，也可以将三种方法混合起来

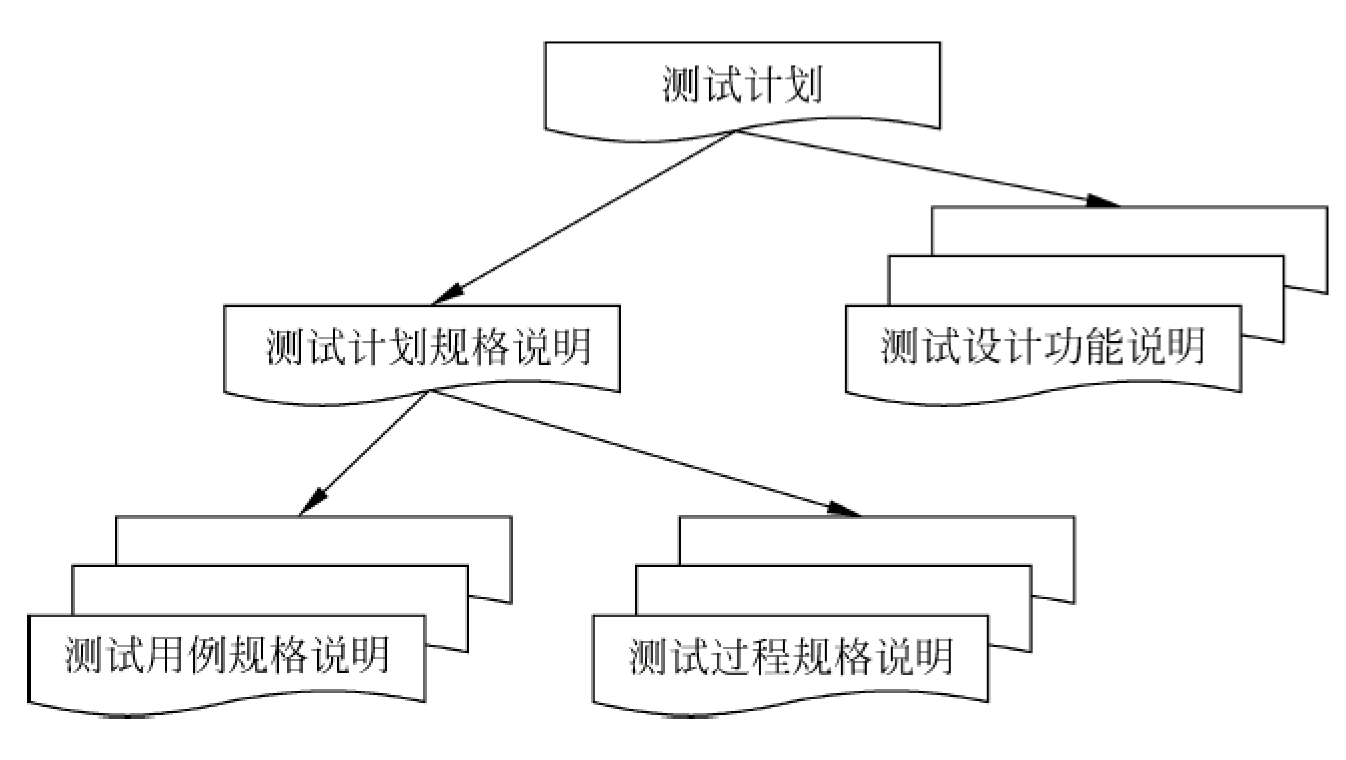


图 5-7　测试用例组织

灵活运用,例如可以先按照不同的程序功能块将测试用例分成若干个模块,然后在不同的模块中划分出不同类型的测试用例,按照优先级顺序进行排列,这样就能形成一个完整而清晰的、基于测试用例进行组织的测试计划。

2. 测试用例的评审

首先要清楚内部评审的定义,是测试组内部的评审,还是项目组内部的评审。评审的定义不同,内容也不会相同。如果是测试组内部的评审,应该着重于:

(1) 测试用例本身的描述是否清晰,是否存在二义性。

(2) 是否考虑到测试用例的执行效率。往往测试用例中步骤不断重复执行,验证点却不同,加之测试设计的冗余性,都造成了效率的低下。

(3) 是否针对需求跟踪矩阵,覆盖了所有的软件需求。

(4) 是否完全遵守了软件需求的规定。这并不是一定的,因为即使再严格的评审,也会出现错误,应具体情况具体对待。

如果是项目组内部的评审,就需要评审委员会来做了,角度不同,评审的标准也不同。比如,收集客户需求的人注重业务逻辑是否正确;分析软件需求规格的人注重用例是否跟规格要求一致;开发负责人会注重用例中对程序的要求是否合理。

表 5-7 是一份测试用例评审检查单。

表 5-7　测试用例评审检查单

序　　号	主要检查项
1	《需求规格说明书》是否评审并建立了基线?
2	是否按照测试计划时间完成测试用例的设计?
3	需求新增和变更是否进行了对应的调整?
4	测试用例是否按照公司定义的模板进行编写?
5	测试用例是否覆盖了《需求规格说明书》?
6	测试用例编号是否和需求进行对应?
7	非功能测试需求或不可测试需求是否在测试用例中列出并说明?
8	测试用例设计是否包含了正面、反面的用例?
9	每个测试用例是否清楚地填写了测试特性、步骤、预期结果?

续表

序 号	主要检查项
10	步骤/输入数据部分是否清晰,是否具备可操作性?
11	测试用例是否包含测试数据、测试数据的生成办法或者输入的相关描述?
12	测试用例是否包含白盒测试?是否针对不同部分使用不同设计方法?
13	重点需求用例设计至少要有三种设计方法,是否满足?
14	每个测试用例是否都阐述预期结果和评估该结果的方法?
15	若需要进行打印是否存在打印位置?
16	用例覆盖率是否达到相应质量指标?

在评审活动中可收集用例的反馈信息,在此基础上进行用例更新,直到通过评审。

3. 测试用例的管理软件

如今,通常使用测试用例管理工具管理测试用例。使用工具对软件的整个测试输入、执行过程和测试结果来进行记录管理,可以提高回归测试的效率、大幅度缩短测试时间,可以提高测试质量、用例复用度、需求覆盖率等。

在没有测试管理工具之前,只能使用 Word 或 Excel 管理测试用例,这会带来不少维护弊端,若有专门的测试用例管理工具,很多问题则可解决。常见的测试用例管理工具有 TestManager、JIRA、Wiki、TestLink 等。表 5-8 列出了这些测试管理工具的比较。

表 5-8 常用测试管理工具的比较

工 具 名	优 点	缺 点
TestManager	① 功能强大 ② 文件夹形式的管理,可以对测试用例无限分级 ③ 可以和 Rational 的测试工具 robot、functional 相结合 ④ 有测试用例执行的功能,但必须先生成对应的手工或自动化脚本	① 本地化支持不好 ② 汉字显示太小 ③ 需安装客户端才可使用,和开发人员交流不方便 ④ 测试用例的展示形式单一
Wiki	① Web 界面形式,交流方便 ② 测试用例展示形式多样,可以贴图,可以进行格式化的编辑 ③ 可为测试用例添加注释,方便测试用例评审 ④ 具有强大的全文搜索功能	① 不是专业的测试用例管理工具 ② 无法和其他测试工具集成 ③ 测试用例统计不方便,需要专门的程序 ④ 没有测试用例执行和跟踪功能 ⑤ 没有定制统一的模板
Bugzilla+ Test Runner	① 开源、免费 ② Web 方式的管理界面 ③ 自动邮件提醒 ④ 和缺陷管理系统 Bugzilla 结合紧密,有测试用例执行管理功能 ⑤ 测试用例可分优先级 ⑥ 测试用例可评审	① 安装设置较烦琐 ② 没有配置过的经验 ③ 测试用例必须按照一个步骤对应一个验证点的形式来编写

续表

工　具　名	优　　点	缺　　点
TestDirector	① 功能强大 ② Web 方式的界面 ③ 有测试用例执行跟踪功能 ④ 有灵活的缺陷定制 ⑤ 和自身的缺陷管理工具紧密集成 ⑥ 界面较友好	① 每个项目库同时在线人数有限制 ② 可能存在部分不稳定性，但是基本功能没有问题
CQ	① 和 CQ 的缺陷管理紧密结合 ② 可以使用 CQ 强大的查询和图表功能	Eclipse 的界面，较为笨重，需要安装
Excel	① 依托 Excel 本身强大的功能 ② 灵活，易于扩展	① 维护比较麻烦 ② 统计、度量不方便
Word	① 依托 Word 本身强大的功能 ② 灵活，易于扩展	① 格式不统一 ② 统计不方便

5.6　本章小结

软件测试的重要性是毋庸置疑的，但如何以最少的人力、资源投入，在最短的时间内完成测试，发现软件系统的缺陷，保证软件的优良品质，是软件公司不懈探索和追求的目标。每个软件产品或软件开发项目都需要有一套优秀的测试方案和测试方法。

测试的类型包括功能测试、边界测试、异常测试、性能测试、压力测试等。在用例设计中，除了功能测试用例外，应尽量考虑边界、异常、性能的情况，以便发现更多的隐藏问题。

根据测试用例的属性，可分阶段、分模块来构造测试套件，以更好地组织和执行测试用例。随着需求的变化，测试用例要做相应的改动；随着测试的深入，测试人员对产品的特性有更深的理解，会发现更多的缺陷，需要不断完善现有的测试用例。也就是说，在整个软件开发周期中要对测试用例进行有效的跟踪和维护。

习题 5

1. 阐述测试用例在测试过程中所起的作用，标准的测试用例有哪几个部分组成？测试用例一般采用哪些方法来进行组织？

2. 如何确定软件的测试范围？哪些因素是要重点考虑的？

3. 一个函数，要求用户输入 6 位正整数，请设计所有测试用例。

4. 按要求给出下图的测试用例。

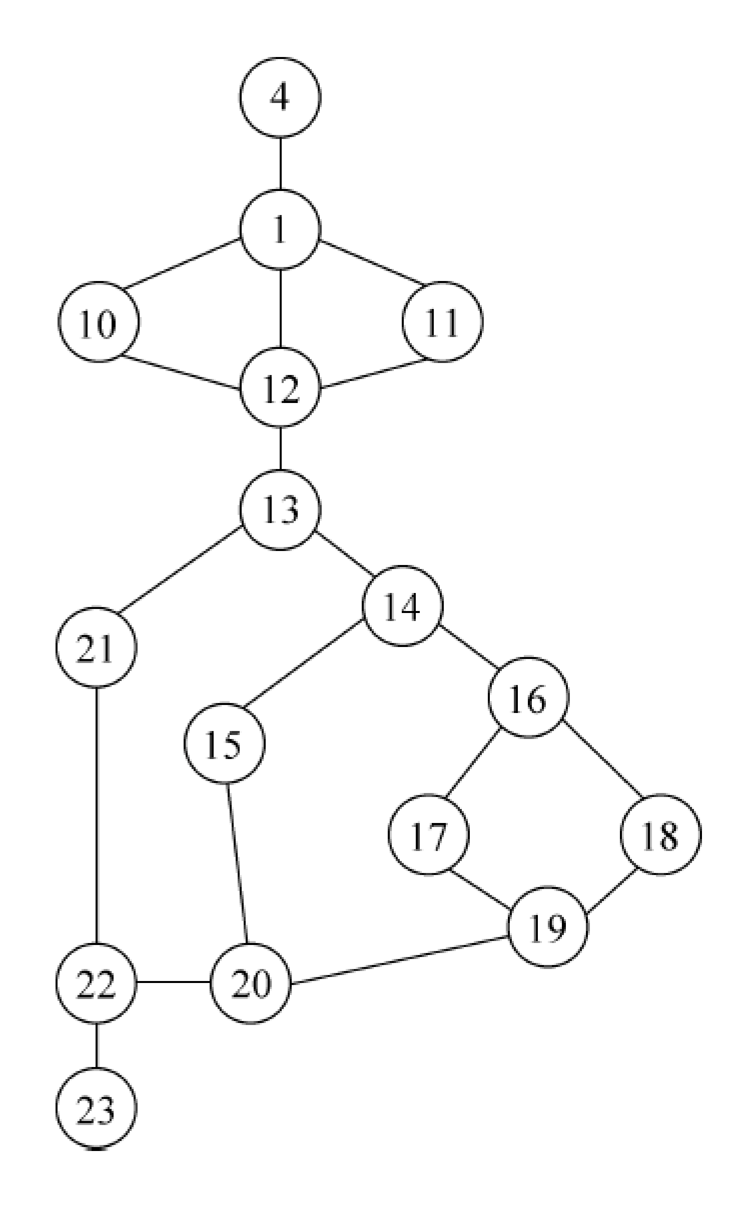
(1) 语句覆盖；

(2) 判定覆盖；

(3) 条件覆盖；

(4) 判定/条件覆盖；

（5）条件组合覆盖。



第6章 测试报告与测试评测

软件测试的目的是在软件开发的过程中，对软件产品进行质量控制，保证软件产品的最终质量。一般来说软件测试应严格按照软件测试流程，制定测试计划、测试方案、测试规范，然后实施测试，对测试数据进行记录，并根据测试情况撰写测试报告。测试报告主要是报告发现的软件缺陷。测试评价主要包括覆盖评价以及质量和性能评价。覆盖评价是对测试完全程度的评测，质量和性能评价是对测试的软件对象的性能、稳定性以及可靠性的评测。

本章主要介绍如何报告发现的软件缺陷，以及有关测试评估和撰写测试总结报告等相关知识。

6.1 软件缺陷和软件缺陷种类

6.1.1 软件缺陷案例

让我们回顾一些"臭名昭著"的软件缺陷案例，它们都是由于软件测试不充分而导致的严重问题。

1963年，由于用FORTRAN程序设计语言编写的飞行控制软件中的循环语句"DO 5 I=1,3"误写为"DO 5 I=1.3"，结果导致美国首次金星探测飞行失败，造成价值1000多万美元的损失。

1979年，新西兰航空公司的一架客机因计算机控制的自动飞行系统发生故障而撞在阿尔卑斯山上，机上257名乘客全部遇难。

1983年，美国科罗拉多河水泛滥，但由于计算机对天气形势预测有误，水库未能及时泄洪，以致造成严重的经济损失和人员伤亡。

1990年1月15日，通信中转系统软件发生故障，导致主干远程网大规模崩溃，使数以千计的电信运营公司损失惨重。

1992年10月26日，伦敦救护中心的计算机辅助发送系统刚启动就崩溃了，导致这个全世界最大的(每天要接运五千多名病人)救护机构全部瘫痪。

1994年，美国迪斯尼公司的"狮子王"软件在少数系统中能正常工作，但在大众使用的常见系统中无法正常运行。后来证实，这是由于迪斯尼公司没有对市场上投入使用的各种PC机型进行正确的测试。同年，英特尔奔腾浮点除法发生软件缺陷，英特尔为处理此软件缺陷支付了4亿多美元。

1996 年 6 月 4 日，欧洲航空航天局耗资 80 亿美元发射的阿里亚娜 501 火箭，在发射升空 37 秒后爆炸。原因是主发动机打火顺序开始 37 秒后，制导信息由于惯性制导系统的软件出现规格和设计错误而完全丢失。

临近 2000 年时，计算机业界一片恐慌，导致这种现象发生的就是著名的“千年虫”问题。而其原因是在 20 世纪 70 年代，由于计算机硬件资源很珍贵，程序员为节约内存资源和硬盘空间，在存储日期数据时，只保留年份的后 2 位，如“1980”被存储为“80”。当 2000 年到来时，问题出现了，计算机无法分清“00”是指“2000 年”还是“1000 年”。例如银行存款的软件在计算利息时，用日期“00”减去当时存款的日期，结果存款年数就变为负数，导致顾客反要付给银行巨额的利息。为了解决“千年虫”问题，大量的人力、物力和财力被消耗。

2008 年，我国举办了首次奥运会。当然，问题发生在 2007 年 10 月 30 日，当日上午 9 时，北京奥运会门票面向境内公众销售第二阶段正式启动，系统访问流量猛增，官方票务网站流量瞬时达到每小时 800 万次，超过了系统设计每小时 100 万次的承受量，奥运门票系统访问量超过原计划的 8 倍，造成网络拥堵，售票速度慢或暂时不能登录系统的情况，直接造成公众无法及时提交购票申请，北京奥运票务中心不得不就此向广大境内公众购票人发布致歉信。

6.1.2 软件缺陷的含义

软件缺陷，即计算机系统或者程序中存在的任何一种破坏正常运行能力的问题、错误或者隐藏的功能缺陷、瑕疵。从产品内部看，软件缺陷是软件产品开发或者维护过程中所存在的错误、毛病等各种问题；从外部看，软件缺陷则是系统所需要实现的某种功能的失效或者违背。

通常认为，符合下面 5 个规则之一的就是软件缺陷。

(1) 软件未达到产品说明书中已经表明的功能；

(2) 软件出现了产品说明书中指明不会出现的错误；

(3) 软件未到达产品说明书中虽未指出但应当达到的目标；

(4) 软件功能超出了产品说明书中指出的范围；

(5) 软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不良。

在软件系统的执行过程中遇到一个软件缺陷，就可能引起软件系统的失效。准确、有效地定义和描述软件缺陷，可以使其得到快速修复，节约软件测试项目的成本和资源，提高软件产品的质量。

软件缺陷的基本描述是报告软件缺陷的基础部分，一个好的描述需要使用简单、准确、专业的语言来抓住软件缺陷的本质。描述的信息含糊不清，则可能会误导开发人员。以下是软件缺陷的一些有效描述规则。

(1) 单一准确：每个报告只针对一个软件缺陷。

(2) 可以再现：提供出现这个缺陷的精确步骤，使开发人员能看懂，可以再现并修复缺陷。

(3) 完整统一：提供完整、前后统一的软件缺陷的修复步骤和信息，如图片、Log 文件等。

(4) 短小简练：通过使用关键词，使软件缺陷的标题描述短小简练，又能准确解释产生

缺陷的现象。

(5) 特定条件：软件缺陷描述不要忽视那些看似细节但又必要的特定条件(如特定的操作系统、浏览器等)。这些特定条件能提供帮助开发人员找到缺陷原因的线索。

(6) 补充完善：从发现软件缺陷开始，测试人员的责任就是保证它被正确地报告，并得到应有的重视，然后继续监视其修复的全过程。

(7) 不作评价：软件缺陷报告是针对软件产品的，因此软件缺陷描述不要带有个人观点，不要对开发人员进行评价。

6.1.3　软件缺陷的种类

在软件测试过程中如何判断软件缺陷？软件缺陷都有哪些种类？资历较浅、经验不足的测试人员对软件缺陷的认定往往会没有把握。这里指出的软件缺陷有15种，包括功能不正常、软件在使用上不方便、软件的结构未做良好规划、功能不充分、与软件操作者的互动不良、使用性能不佳、未做好错误处理、边界错误、计算错误、使用一段时间所产生的错误、控制流程的错误、在大数据量压力之下所产生的错误、不同硬件设备所导致的错误、版本控制不良所产生的错误和软件文档错误。下面分别进行说明。

1. 功能不正常

简单地说就是所提供的功能，在使用上并不符合设计规格说明书中规定的功能，或者根本无法使用。这个错误常常会发生在测试过程的初期和中期，有许多在设计规格说明书中规定的功能无法运行，或是运行结果不符合预期设计。最明显的例子就是在用户接口上所提供的选项及动作，使用者在操作后毫无反应。例如，有一个简单的软件测试实例，软件让使用者输入所想要保留信息的数量及天数，测试人员发现输入数量没有问题，但是输入天数则没有作用。经过查证之后发现，开发人员忘记加入输入选项的判断式，导致无论使用者选择什么输入，程序都是以输入保留信息的数量进行运算。这对开发人员来说是一个很简单的小问题，可是如果测试人员未做好把关工作，这个问题对用户来说就是一个很大的错误。

2. 软件在使用上不方便

只要是不知如何使用或难以使用的软件，在设计上一定是出了问题。所谓好用的软件就是使用上尽量方便，使用户易于操作。例如，微软所推出的软件，在用户接口及使用操作上确实是下了一番工夫。有许多软件公司推出的软件，在彼此的使用接口上完全不同，这样的做法其实只会增加使用者的学习难度，另一方面也凸显了这些软件公司集成能力的不足。

3. 软件的结构未做良好规划

这里的结构主要指的是软件是自顶向下的方式开发，还是自底向上的方式开发。以自顶向下的结构或方法所开发的软件，其功能的规划及组织会比较完整。相反，以自底向上的方式开发出来的软件功能则较为分散。举例来说，假设有一个软件提供3个扫描的功能：实时扫描、手动扫描和扫描。就功能方面而言，这3种功能应该放到同一个扫描选项内。若实时扫描是后来增加的，而且提供了立即编辑的功能，则实时扫描应被独立出来成为另一个

单独选项。否则，会使许多使用者误以为在实时扫描所做的立即编辑设置，适用于其他两种扫描功能。

4. 所提供的功能不充分

这个问题与功能不正常是不一样的。这里所指的是软件所提供的功能在运作上是正常的，可是对使用者而言却是不完整的。即使软件的功能运作结果符合设计规格的要求，系统测试人员在测试结果的判断上，也一定要从使用者的角度进行思考。这里举一个例子，假设所测试的软件提供了数据处理功能，但是采用的是封闭式的 CodeBase 数据库。对开发人员来说，采用 CodeBase 数据库对程序编写来说比较容易，经过测试之后也未发生其他的问题。可是在用户的环境下进行测试之后才发现，用户要求提供支持 SQL 数据库的功能，因为他们希望能够统一管理所有的信息。在这种情况下，系统测试人员必须将这个问题呈现出来，虽然现在要求增加这个需求已经太晚了，不过可以建议提供另一种解决方法，例如提供一个信息转换工具等。测试人员要随时对所进行测试的功能保持一个存疑的态度，因为这样的问题如果出现在开发的后期，所能提供的解决方式就很有限，所以早一点发现这样的问题对提高整个开发质量的帮助很大。通常这样的问题大都是由经验较丰富的测试人员发现的。

5. 与软件操作者的互动不良

一个好的软件必须与操作者正常互动。在操作者使用软件的过程中，软件必须能够很好地响应操作者。例如，在网络中浏览网页时，假设操作者在某一个网页填写信息，但是所填写的信息不足或是有误。当操作者单击了“确定”按钮之后，网页此时响应“操作者所填写的信息有错”，可是并未指出错误在哪里，操作者只好回到上一页后重新填写一次，或是直接放弃离开。这个问题的出现就是因为软件在操作互动方面未做完整的设计。对于窗口类型的软件，这一点也常常被忽略，例如当操作者做任何更新或删除动作，软件是否提供相应的信息给使用者？或要求对所执行的动作做确认(如提供确认窗口等)？与操作者的互动原则就是，所有的动作必须伴随着适当的响应。

6. 使用性能不佳

所测试的软件功能正常，但是使用性能不佳，这样算不算问题呢？使用性能不佳，当然是一个问题。这样的问题通常是开发人员采用了错误的解决方案，或是运用了不适用的算法所导致的。在实际测试中发现有不少错误都是因为采用了错误的解决方法。例如有一个软件属于 Client/Server 的企业软件，Server 端将 Client 传递上来的信息做分类处理。由于信息包含的种类相当多，于是开发人员将它们分别存入不同的信息文件内。例如，Client A 送给 Server 的信息的种类有 $A_1 \sim A_{10}$，Server 分别将信息存到 10 不同的信息文件内。这样造成使用者查询信息时速度很慢，因为 Server 会逐一查找 10 个不同的信息文件来做对比。类似的例子相当多，寻根究底是因为未做好基础审核及设计审核，直到进行系统测试或性能测试时才凸显出问题的严重性。

7. 未做好错误处理

软件除了要避免出错之外，还要做好错误处理，许多软件之所以会产生错误是因为程序

本身不知道如何处理所遇到的错误。例如，所测试的软件需要读取外部的信息文件并进行一些分类整理，可是刚好被读取的外部信息文件的内容已被损毁。当程序读取这个损毁的信息文件时，程序发现问题，这时候操作系统不知如何处理这个状况，为了保护自己只好中断程序。由此可见，应设立错误处理机制。如上述的例子，程序在读取外部信息文件之前，应该先检查外部信息文件是否损毁，这样才比较保险。

8. 边界错误

缓冲区溢出在这几年已成为网络攻击的常用方式，而这个错误就属于边界错误的一种。简单地说，边界错误即程序本身无法处理超越边界所导致的错误。编程语言所提供的函数有问题是造成边界错误的一大原因，除此以外，许多情形是开发人员在声明变量或是使用边界范围时不小心引起的。

9. 计算错误

只要是计算机程序，就免不了包含数学计算。软件之所以会出现计算错误，大部分的原因在于采用了错误的数学运算公式或未将累加器初始化为0。

10. 使用一段时间所产生的错误

程序刚开始运行时很正常，但在运行了一段时间后功能却出现问题。最典型的例子就是数据库的查找功能。有一些软件在刚开始使用时，所提供的信息查找功能运作良好，可是在使用了一段时间后却发现，进行信息查找所需的时间越来越长了。原因在于，程序采用的信息查找方式是顺序查找，随着数据库信息量的增加，查找的时间当然会越来越长。还有一个例子，一个软件提供组件更新的功能，程序会通过因特网来下载最新的组件，之后程序会以新的组件取代旧的组件。但开发人员在设计中忘了将状态标识恢复到原来的状态，结果这个更新程序做第一次更新动作的时候正确运作，可是再做第二次更新动作就毫无作用了。

11. 控制流程的错误

控制流程的好与坏，考验着开发人员对软件开发的态度，关系到设计的程序是否严谨。软件状态间的转变是否合理，要依据流程进行控制。用软件安装程序来解释这样的问题是最容易理解的。比如在进行软件安装时，用户输入用户名及一些简单信息后，软件就直接进行安装了，可是安装的磁盘驱动器或目录却不是用户所希望的，问题就出在安装程序并未向用户提供可以更改安装目的地的选择。这就是软件控制流程不完整的错误问题。

12. 在大数据量压力之下所产生的错误

程序在大数据量状态下运作不正常，就属于这种软件错误。大数据量压力测试对于Sever级的软件是必须要进行的一项测试，因为Sever级的软件对稳定度的要求远比其他软件高。通常，一连串的大数据量压力测试是必须实施的。例如，让程序处理超过10万次的信息，然后再来观察程序运行的结果。

13. 在不同硬件环境下产生的错误

顾名思义，就是问题的产生与硬件环境的不同有关。如果所开发的软件与硬件设备有直接的关系，这样的问题就会相当多。例如，有的软件在某些品牌的服务器上运行时就会出错。

14. 版本控制不良所产生的错误

这样的问题属于项目管理的疏忽，当然测试人员未善尽职守也是原因之一。例如，一个软件被反映有安全上的漏洞，后来软件公司也很快将这个问题的修改版提供给用户。但是在一年后他们在推出新版本时，却忘记将这个已解决掉的问题加入新版本内。所以对用户来说，原来的问题已经解决了，可是想不到将版本升级之后，问题却又出现了。试问这些用户会如何看待这个软件的质量？其实这样的问题发生的概率不小。有一些用户对这种情况所采取的态度就是，尽量不去变更已经相当稳定的软件，就算软件公司提供新的版本或修正版，他们也是采取先按兵不动的方法。这个方式虽然安全但是也会带来新的危机，因为软件始终可能存在未知错误或缺陷。2003 年 1 月 24 日，针对微软公司 SQL Sever 漏洞的计算机蠕虫病毒导致了大规模的计算机网络瘫痪。而之所以会造成这样大的伤害，就是因为有许多用户没有及时安装微软公司所提供的补丁程序。

15. 软件文档的错误

最后这个错误是软件文档错误。这里的文档错误除了软件所附带的使用手册、说明文档以及其他相关的软件文档内容错误之外，还包括了软件使用接口上的错误文字和错误用语。错误的软件文档内容除了降低质量之外，最主要的问题是会误导用户。

以上这些软件缺陷类型，必须通过软件测试工作来仔细识别。

6.1.4 软件缺陷的严重性等级

软件缺陷一旦被发现，就要设法找出引起这个缺陷的原因，分析对产品质量的影响，然后确定软件缺陷的严重性和处理这个缺陷的优先级。各种软件缺陷所造成的后果是不同的，有的仅仅是带来不方便，有的则可能是灾难性的。一般来说，问题越严重的，其优先级越高，越要得到及时的纠正。软件公司对缺陷严重性级别的定义不尽相同，按照 CMM5 中定义的规范，软件缺陷分为四个等级，分别为致命、严重、一般和提示。

1. 致命

致命性漏洞主要为：内存泄漏、用户数据丢失或被破坏、系统崩溃/死机/冻结、模块无法启动或异常退出、严重的数值计算错误、功能设计与需求严重不符，以及其他导致无法测试的错误和造成系统不稳定等。

2. 严重

严重性漏洞主要为：功能未实现、功能严重错误、系统刷新错误、语音或数据通信错误、轻微的数值计算错误、系统所提供的功能或者服务受到明显的影响但不会影响到系统稳

定性。

3. 一般

一般性漏洞主要为：操作界面错误(包括数据窗口内列名定义、含义不一致)、边界条件下错误、提示信息错误(包括未给出信息、信息提示错误等)、长时间操作无进度提示、系统未优化(性能问题)、光标跳转设置不好、鼠标(光标)定位错误等。

4. 提示

提示性漏洞主要为：界面格式等不规范、辅助说明描述不清楚、操作时未给出用户提示、可输入区域和只读区域没有明显的区分标志、个别不影响产品理解的错别字、文字排列不整齐等一些小问题及建议。

6.2 软件缺陷的生命周期

生命周期的概念来自于一个物种从诞生到消亡所经历的过程，软件缺陷也经历这样的过程。当一个软件缺陷被发现并报告出来之时，意味着这个缺陷诞生了。缺陷被修正之后，经过测试人员的进一步验证，确认这个缺陷不复存在，然后测试人员关闭这个缺陷，意味着缺陷走完它的历程，结束其生命周期。可以看出，缺陷的生命周期可以简单地表现为“发现—打开—修正—关闭”。

软件缺陷生命周期各阶段简述如下。

(1) 发现——打开：测试人员找到软件缺陷并将软件缺陷提交给开发人员；

(2) 打开——修复：开发人员再现、修复缺陷，然后提交测试人员去验证；

(3) 修复——关闭：测试人员验证修复过的软件，关闭已不存在的缺陷。

但是这是一种理想的状态，在实际的工作中是很难这样顺利的，需要考虑的各种情况还是非常多的。例如，不是每个缺陷都能及时得到修正，可能由于时间关系或技术限制，某些缺陷不得不延迟到下一个版本中去修正。有些缺陷描述不清楚，开发人员看不懂或不能再现，将缺陷打回，让测试人员补充信息。有些缺陷得到了开发人员处理，认为已得到修正，测试人员验证之后，发现缺陷依旧存在，没有得到彻底的处理。这样，测试人员不得不重新打开这个缺陷，交给开发人员去处理。由此可见，在有些情况下，生命周期变得更复杂一些。图 6-1 所示就是复杂但更契合实际的软件缺陷生命周期。

通常，软件缺陷生命周期有如下两个附加状态。

1. 审查状态

在审查状态项目管理员或者委员会会决定软件缺陷是否应该修复。在某些项目中，这个过程直到项目将结束时才发生，甚至根本不发生。注意，从审查状态可以直接进入关闭状态。如果审查发现软件缺陷太小，认定软件缺陷不应该修复、不是真正的问题或者属于测试失误，软件缺陷就会进入关闭状态。

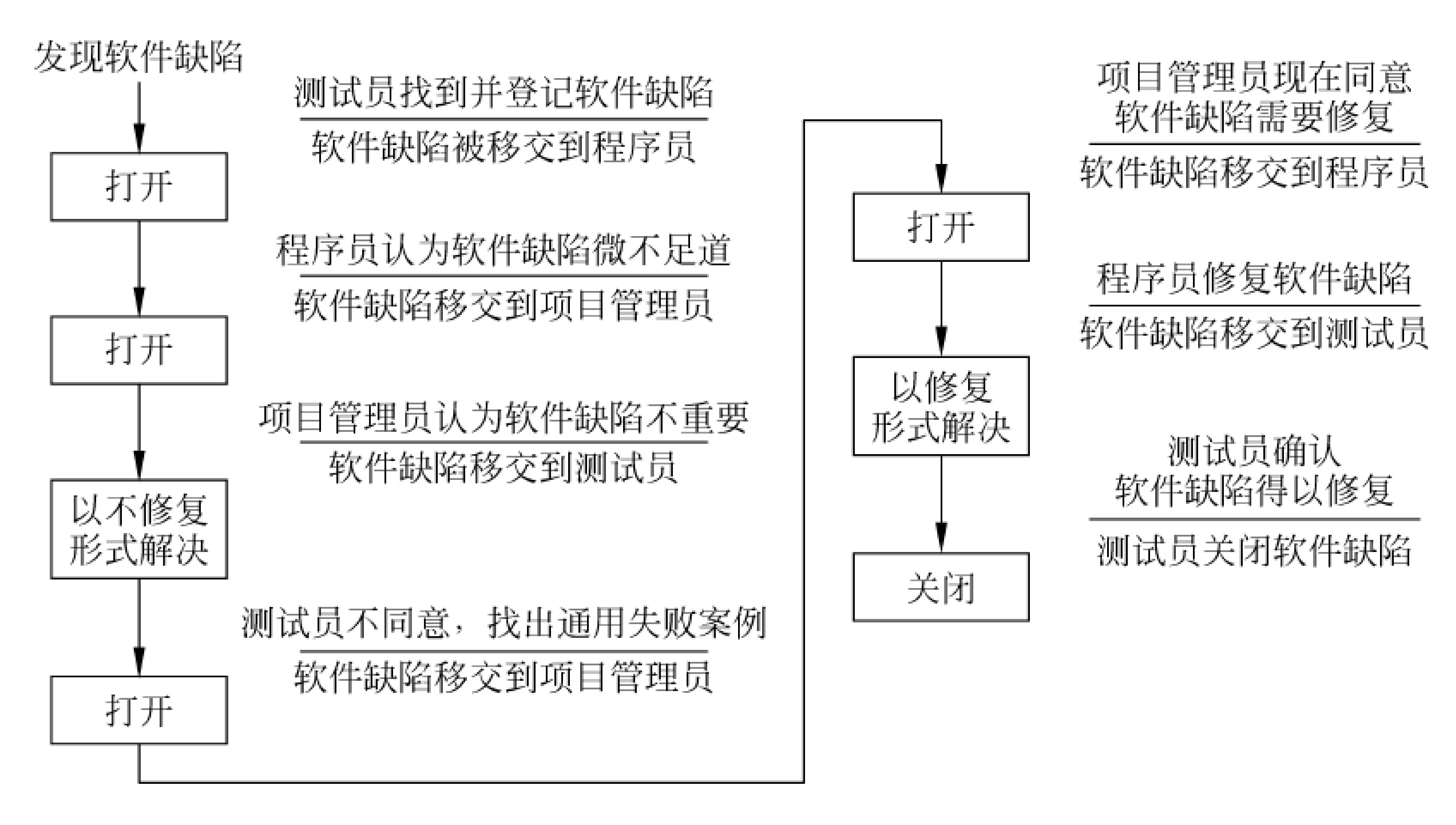


图 6-1 复杂的软件缺陷生命周期

2. 推迟状态

审查可能认定软件缺陷应该在将来的某一时间考虑修复，但是在该版本软件中可不修复。

推迟修复的软件缺陷以后也可能被证实很严重，要立即修复。此时，软件缺陷会被重新打开，再次启动整个过程。

大多数项目小组采用规则来约束由谁来改变软件缺陷的状态，或者交给其他人来处理软件缺陷。例如，只有项目管理员可以决定推迟软件缺陷修复，或者只有测试人员允许关闭软件缺陷。重要的是一旦软件缺陷登记了，就要跟踪其生命周期，不能跟丢了，并且应提供必要信息驱使其得到修复和关闭。

综上所述，软件缺陷在生命周期中可能经历数次的审阅和状态变化，最终由测试人员关闭来结束其生命周期。软件缺陷生命周期中的不同阶段应是测试人员、开发人员和管理人员一起参与、协同测试的过程。软件缺陷一旦发现，便进入测试人员、开发人员、管理人员的严密监控之中，直至软件缺陷生命周期终结，这样可保证在短时间内高效率地关闭所有的缺陷，缩短软件测试的进程，提高软件质量，同时减少开发和维护成本。

6.3 分离和再现软件缺陷

测试人员要想有效报告软件缺陷，就要对软件缺陷以明显、通用和再现的形式进行描述。

分离和再现软件缺陷是考验软件测试人员专业技能的地方，测试人员应该设法找出缩小问题范围的具体步骤。对测试人员有利的情况是，若建立起绝对相同的输入条件时，软件缺陷就会再次出现，不存在随机的软件缺陷。

如果找到的软件缺陷要采取繁杂的步骤才能再现，或者根本无法再现，碰到这种情况，可采取如下的方法来分离和再现软件缺陷。实践证明这些方法对测试人员是有所帮助的。

1. 记录所有的步骤

记下所做的每一件事——每一个步骤、每一次停顿、每一件工作。所有这些的目的是确

保导致软件缺陷所需的全部细节可见，并且可以从另一个角度分析。

2. 查找时间依赖和竞争条件的问题

软件缺陷仅在特定时刻出现吗？也许它取决于输入数据的速度，或者正使用慢速软盘代替快速硬盘保存数据。看到软件缺陷时网络忙吗？在更慢或更快的硬件上运行测试用例，要考虑时序。

3. 注意边界条件软件缺陷、内存泄漏和数据溢出等问题

比如执行某个测试可能导致数据覆盖，但是也只有在试图使用这些数据时才会发现。重新启动计算后消失，而仅在执行其他测试之后出现的软件缺陷常常属于这一类。如果发生这种现象，就要查看前面执行的测试，也许要利用一些动态白盒技术，看软件缺陷是否在无意间发生了。

4. 状态缺陷仅在特定软件状态中显露出来

状态缺陷的例子是软件缺陷仅在软件第一次运行时出现或者在第一次运行之后出现。软件缺陷也许仅在保存数据之后，或者按任何键之前发生。状态缺陷看起来很像时间依赖和竞争条件的问题，但是你会发现时间并不重要——重要的是事件发生的次序，而不是发生的时间。

5. 考虑资源依赖性和内存、网络、硬件共享的相互作用

软件缺陷是否仅在运行其他软件并与其他硬件通信的“繁忙”系统上出现？虽然软件缺陷可能最终会被证实是竞争条件、内存泄漏或者状态缺陷由于软件的依赖性或者对资源的相互作用而进一步恶化，但是查一查这些影响有利于分离软件缺陷。

6. 不要忽视硬件

与软件不同，硬件可能降级，不按预定方式工作。板卡松动、内存条损坏或者 CPU 过热都可能导致像是软件缺陷的失败，但是事实上不是。设法在不同硬件上再现软件缺陷。这在执行配置和兼容性测试中尤其重要。需要知道的是缺陷是在一个系统上还是在多个系统上出现。

如果尽最大的努力分离软件缺陷，也无法用简明的步骤再现，那么仍然需要记录软件缺陷，以免跟丢了。也许程序员仅用从测试员那里了解到的信息就能够找出问题所在。由于程序员熟悉代码，因此看到症状、测试用例步骤，特别是努力分离问题的过程时，可能得到查找软件缺陷的线索。当然，程序员并不愿意，也没必要对发现的每一个软件缺陷都这样做，但是有时这些难以分离的问题需要小组的共同努力。

6.4 正确面对软件缺陷

在软件测试过程中，测试人员必须确保测试过程发现的软件缺陷得到关闭。测试是为了证明程序有错，而不是证明程序没错。不管测试计划多么完善和执行测试多么努力，也不

能保证所有软件缺陷发现了就能修复。有些软件缺陷可能会完全被忽略，还有一些可能推迟到软件后续版本中修复。

在实际测试工作中，软件测试人员需要从综合的角度考虑软件的质量问题，对找出的软件缺陷保持一种平常心态。

1. 并不是测试人员辛苦找出的每个软件缺陷都是必须修复的

不修复软件缺陷的原因可能如下：

(1) 没有足够的时间。

在任何一个项目中，软件功能通常都较多，而程序设计人员和软件测试人员较少，并且可能在项目进度中没有为编制和测试留出足够的时间。在实际开发过程中，经常出现用户对软件的完成提出一个最后期限，在最后期限之前，必须按时完成软件。

(2) 不算真正的软件缺陷。

在某些特殊场合，错误理解、测试错误或者说明书变更，使得一些"软件缺陷"可不当作缺陷来对待。

(3) 修复的风险太大。

这种情形比较常见。软件本身是脆弱的，难以理清头绪，有点像一团乱麻。修复一个软件缺陷可能导致其他软件缺陷出现。在紧迫的产品发布进度压力之下，修改软件缺陷将冒很大的风险。不去理睬已知的软件缺陷，以避免出现未知新缺陷的做法也许是安全的办法。

(4) 不值得修复。

虽然有些不中听，但这却是真实的。不常出现的软件缺陷和在不常用功能中出现的软件缺陷可以放过。如果缺陷可以躲过，或者用户有办法预防，这样的软件缺陷通常不用修复。这些都要归结为商业风险决策。

2. 发现的缺陷数量说明不了软件的质量

软件中不可能没有要缺陷，发现很多的缺陷对于测试工作来说，是件很正常的事。缺陷的数量大，只能说明测试的方法很好，思路很全面，测试工作有成效。但是，以此来否认软件的质量，则比较武断。

例如，如果测试中发现的缺陷绝大多数都是提示性错误、文字错误等，错误的等级很低，对它的修改几乎不会影响到执行指令的部分，而关于软件的基本功能或者是性能只发现很少的缺陷。很多时候，这样的测试实际上证明了"软件的质量是稳定的"，因而它属于优秀软件的范畴。这样的软件，只要处理好发现的缺陷，基本就可以发行使用了。对这样的软件进行完整的回归，反而增加软件的成本，浪费商机和时间。

反过来，如果在测试中发现的缺陷比较少，但是这些缺陷都属于功能没有实现、性能没有达标，或经常死机、系统崩溃等现象，而且大多数用户在使用过程中都会发现类似问题。这样的软件不会有人轻言"发布"，因为要承担的风险太大了。

虽然，这两个例子都比较极端，在实际的测试中，几乎不会发生，但是，还是希望测试人员不要把工作集中在发现缺陷的数量上。

3. 不要指望找出软件中的所有缺陷

很多人都知道这个道理，但是却不明白这个规则对软件测试工作的意义。软件中的缺

陷既然是不可能全部发现的，就不要指望找出软件中全部的缺陷，当它足够少（各公司的定义是不同的）的时候，就应该停止测试了。

虽然软件测试人员需要对自己找出的软件缺陷保持一种平常心态，但同时又必须坚持有始有终的原则，跟踪每一个软件缺陷的处理结果，确保软件缺陷得到关闭。关闭软件缺陷的前提是缺陷得以修复，或决定不做修复。缺陷是否需要修复的最终决定权在软件的项目负责人，但将缺陷关闭的责任却在测试人员。

6.5　报告软件缺陷

6.5.1　报告软件缺陷的基本原则

在软件测试过程中若发现软件缺陷，测试人员应及时、清晰地把问题报告给负责判断是否进行修复的小组，修复小组得到所需要的全部信息，然后才能决定怎么做。但是，由于软件开发模式不同和修复小组的不固定性，将这样的决定过程运用于每一个具体小组或者项目是不可能的。在许多情况下，决定权在项目管理员手上。还有一些情况，决定权在程序员手里，也可能留在会议上决定。一般地，由一些专门人员或者团队来审查发现的软件缺陷，判定是否修复。但是，无论什么情况，提供软件缺陷描述的信息对于做决定是十分重要的。若软件测试人员对软件缺陷描述不清楚，报告不够及时、有效，没有建立足够强大的用例来证明指定的软件缺陷必须修复，其结果可能导致对软件缺陷的判断失误，认为软件缺陷不够严重，不值得修复等。报告软件测试错误的目的是保证修复错误的人员可以重复报告的错误，从而有利于分析错误产生的原因，定位错误，然后修正。因此，报告软件测试错误的基本要求是准确、简洁、完整、规范。测试人员在报告软件缺陷方面要狠下工夫。

报告软件缺陷的基本原则如下。

1. 尽快报告软件缺陷

软件缺陷发现得越早，留下的修复时间就越多。例如，在软件发布之前几个月就从帮助软件文档中找出错别字，该错误被修复的可能性就很高。

2. 有效地描述软件缺陷

对测试案例、测试过程要准确、有效地描述。其要求如下：

（1）简单与短小。通过使用关键词，可以使软件缺陷的标题的描述短小简练，又能准确解释产生缺陷的现象。如“主页的导航栏在低分辨率下显示不整齐”中“主页”、“导航栏”、“分辨率”等都是关键词。

（2）明确指明错误类型。如功能错误、字节错误等。

（3）单一。一个报告只针对一个软件缺陷，在一个报告中报告多个软件缺陷会导致：只有其中一个软件缺陷被注意和修正，而该报告中其他的缺陷未能得到修正。

（4）使用IT业界惯用的表达术语和表达方法。

以上概括了报告测试错误的规范要求，测试人员应该牢记上面这些关于报告软件缺陷的原则。这些原则几乎可以运用到任何交流活动中，尽管有时难以做到，然而，如果希望有

效地报告软件缺陷，并使其得以修复，这些是测试人员要遵循的基本原则。随着测试者经验的积累，逐渐养成良好的专业习惯，还可不断补充新的规范书写要求。此外，经常阅读、学习高级测试工程师的测试错误报告，结合自己以前的测试错误报告进行对比和思考，可以不断提高技巧。

6.5.2 有效地报告软件缺陷带来的好处

概括起来，有效的软件缺陷描述会帮助我们达到下列目标：

1. 加快缺陷的修正

精准的缺陷描述可极大地帮助开发人员理解问题的所在，容易再现所报告的问题，多数情况下，只有再现软件缺陷，开发人员才能确定导致缺陷产生的根本原因，从而修正缺陷。

2. 提高工作效率

有效的缺陷会加快开发人员对问题的重现和修复，这样每一个缺陷能得到及时的处理，使每一个小组能够有效地工作。

3. 提高测试人员的信任度

有利于开发团队和测试团队之间的沟通和合作，如开发人员对缺陷的响应会改进。

4. 客观、准确的产品质量评估

记下缺陷，可以了解各个模块、各个功能特性存在哪些缺陷，从而对产品(包括各个阶段性产品)质量有一个客观的评价。

5. 预防缺陷

通过所记录的缺陷分析，获得经验和教训，有助于在今后的软件开发中预防缺陷。

6.5.3 IEEE 软件缺陷报告模板

ANS/IEEE 829—1998 标准定义了一个称为软件缺陷报告的文档，用于报告在测试过程期间发生的任何异常事件。简言之，就是用于登记软件缺陷。模板标准如图 6-2 所示，可作为报告软件缺陷的参考。

1. 软件缺陷报告标识符

指定软件缺陷报告的唯一 ID，用于定位和引用。

2. 软件缺陷总结

简明扼要地陈述事实，总结软件缺陷。这里将给出所测试软件的版本引用信息、相关的测试用例和测试说明等信息。对于任何已确定的软件缺陷，都要给出相关的测试用例，如果

某一个软件缺陷是意外发现的，也应该编写一个能发现这个意外缺陷的测试用例。

3. 软件缺陷推述

软件缺陷报告的编写人员应该在报告中提供足够多的信息，使一般修复人员能够理解和再现事件的发生过程。下面是软件缺陷描述中的各个内容。

- 输入

描述实际测试时采用的输入（例如文件、按键等）。

IEEE 标准 829—1998 软件测试文档编制标准
软件缺陷报告模板

目录

1. 软件缺陷报告标识符
2. 软件缺陷总结
3. 软件缺陷描述
   3.1　输入
   3.2　期望得到的结果
   3.3　实际结果
   3.4　异常情况
   3.5　日期和时间
   3.6　规程步骤
   3.7　测试环境
   3.8　再现尝试
   3.9　测试人员
   3.10　见证人
4. 影响

图 6-2　IEEE 软件缺陷报告模板

- 期望得到的结果

即发生事件时，正在运行的测试用例的设计结果。

- 实际结果

将实际运行结果记录在这里。

- 异常情况

指的是实际结果与预期结果的差异有多大。也记录一些其他数据（如果这数据显得非常重要的话），例如有关系统数据量过小或者过大、一月的最后一天等。

- 日期和时间

软件缺陷发生的日期和时间。

- 规程步骤

软件缺陷发生的步骤。如果使用的是很长的、复杂的测试规程，这一项就特别重要。

- 测试环境

所采用的环境。例如，系统测试环境、验收测试环境、客户的测试环境以及测试场所等。

- 再现尝试

为了再现这次测试，做了多少次尝试。

- 测试人员

进行这次测试的人员情况。

- 见证人

了解此次测试的其他人员情况。

4. 影响

软件缺陷报告的“影响”是指软件缺陷对用户造成的潜在影响。在报告软件缺陷时，测试人员要对软件缺陷分类，以简明扼要的方式指出其影响。经常使用的方法是给软件缺陷划分严重性和优先级。当然，具体方法各个公司不尽相同，但是通用原则是一样的。测试实际经验表明，虽然可能永远都不能彻底克服在确定严重性和优先级过程中所存在的不精确性，但是通过在定义等级过程中对较小、较大和严重等主要特征进行描述，完全可以把这种不精确性减小到一定程度。

6.6 软件缺陷的跟踪管理

软件缺陷跟踪管理是测试工作的一个重要部分，测试的目的是为了尽早发现软件系统中的缺陷，因此，对软件缺陷进行跟踪管理，确保每个被发现的软件缺陷都能够及时得到处理是测试工作的一项重要内容。

6.6.1 软件缺陷跟踪管理的目标

软件测试过程简单说就是围绕软件缺陷进行的，对软件缺陷的跟踪管理一般而言需要达到以下的目标：确保每个被发现的软件缺陷都能够被解决——这里解决的意思不一定是被修正，也可能是其他处理方式（例如，在下一个版本中修正或是不修正），总之，对每个被发现的 Bug 的处理方式必须能够在开发组织中达到一致；收集软件缺陷数据并根据软件缺陷趋势曲线识别测试过程的阶段——决定测试过程是否结束有很多种方式，通过软件缺陷趋势曲线来确定测试过程是否结束是常用并且较为有效的一种方式。在一个运行良好的组织中，软件缺陷数据的收集和分析是很重要的，从软件缺陷数据中可以得到很多与软件质量相关的数据。

6.6.2 软件缺陷的描述

1. 对软件缺陷的描述应该包含可追踪信息

如给每个软件缺陷分配一个缺陷号。每个编号必须是唯一的，可以根据该编号搜索、查看该缺陷的处理情况。

2. 对软件缺陷的描述应该包含软件缺陷的基本信息

通常软件缺陷的基本信息包括缺陷状态、缺陷标题、缺陷严重程度、缺陷紧急程度、缺陷提交人、缺陷提交日期、缺陷所属、缺陷解决人、缺陷解决时间、缺陷解决结果、缺陷处理人、

缺陷处理最终时间、缺陷处理结果、缺陷确认人、缺陷确认时间、缺陷确认结果等。

- 缺陷状态：标注缺陷待修正、待评审、待验证、关闭等状态信息。
- 缺陷标题：简明地说明缺陷的类型及内容。
- 缺陷严重程度：测试人员给出的缺陷严重程度估计，可以是致命的、严重的、一般的、提示的。
- 缺陷紧急程度：测试人员给出的测试处理优先级。
- 缺陷提交人：发现此缺陷的测试人员，最好附有联系方式，以方便缺陷处理人员进行确认。
- 缺陷提交日期：提交人提交缺陷的日期。
- 缺陷所属：指缺陷所在的模块或者是缺陷所属的开发文档的名称。
- 缺陷解决人：由谁来进行缺陷的解决，明确是需求分析人员、设计人员还是程序编码人员。
- 缺陷解决时间：项目组负责人返回的缺陷预计处理的时间。
- 缺陷解决结果：预计缺陷修改后能达到的结果。
- 缺陷处理人：应该由谁来处理这个缺陷。
- 缺陷处理最终时间：指缺陷得到处理的实际时间。
- 缺陷处理结果：缺陷最后的实际处理结果。
- 缺陷确认人：由谁来确认缺陷已经得到了修正。
- 缺陷确认时间：缺陷修复的确认工作完成的时间。
- 缺陷确认结果：确认软件缺陷的修正工作是否有效。

6.6.3　软件缺陷管理的一般流程

1. 软件缺陷管理流程图

在开发测试活动中，通常的软件缺陷管理流程如图 6-3 所示。

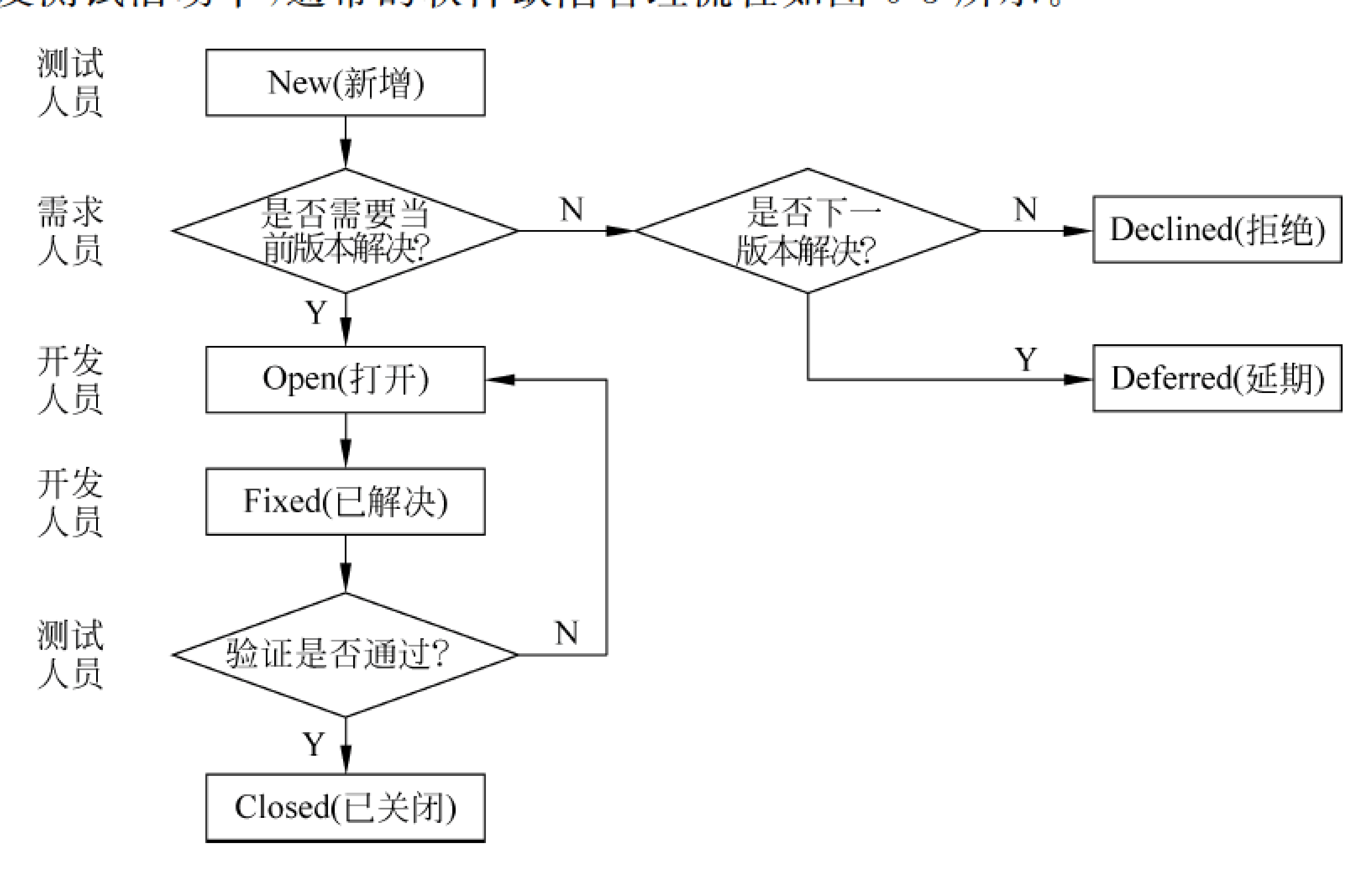


图 6-3　软件缺陷管理流程图

2. 流程图中相关的角色

(1) 测试工程师

在这里主要是指发现和报告软件缺陷的测试人员。在一般流程中,他需要对这个软件缺陷后续相关的状态负责,包括相关人员对这个软件缺陷相关信息的询问回答以及对 Bug 的验证测试。

(2) 开发工程师

这里主要指对这个软件缺陷进行研究和修改的开发人员。同时,对于修改后的软件缺陷在提交给测试人员进行正式测试验证之前,开发工程师还需要对这个软件缺陷进行验证测试。

(3) 需求工程师

主要工作是对软件缺陷进行确认,是否需要解决、是否需要在当前版本解决。

3. 软件缺陷状态的含义解释

(1) New(新缺陷):软件中新发现报告的软件缺陷一般由测试人员提交。当然也可能是开发人员自己在单元或代码测试过程中提交,或从软件使用的最终用户或测试现场反馈得到软件缺陷报告。

(2) Open(打开):处于这个状态时,软件缺陷已经被确认并已经分配给相关的开发人员进行相关的修改。

(3) Fixed(已修改):开发人员将相应的 Bug 修改为 fixed 状态,交付给相关的测试小组进行验证测试。

(4) Closed(结束):测试小组人员对软件缺陷进行验证,通过后将软件缺陷状态改为 closed 状态。

上面介绍的是缺陷管理过程中主要的缺陷状态,或者是软件缺陷处理顺利时所经历的状态。实际上,软件缺陷还有其他一些状态或者可以认为是辅助的状态,分别是:

(1) Declined(拒绝)。需求人员进行分析后认为不是软件缺陷,或通过开发人员的调查研究认为不是软件缺陷。开发人员可以将具体的理由加入到软件缺陷描述或备注中,测试人员验证确实不存在此问题时可以直接将 Bug 关闭。

(2) Deferred(延期)。需求人员确认软件缺陷不在当前版本解决时的状态,在软件缺陷跟踪中可以通过修改软件缺陷的修改预期时间进行延期,需要在备注中说明延期的原因。

6.6.4 软件缺陷数据统计

如前所述,软件缺陷数据统计也是软件缺陷跟踪管理系统的目标。一般而言,生成的软件缺陷数据统计图表包括软件缺陷趋势图、软件缺陷分布图、软件缺陷及时处理情况统计表等。

6.6.5 软件缺陷跟踪管理系统

软件缺陷报告过程是很复杂的,需要大量的信息、详尽的细节和很好的组织工作,才能

有所成效。在实际的软件测试工作中,为了更高效地记录发现的软件缺陷,并在软件缺陷的整个生命周期中对其进行监控,常常运用软件缺陷跟踪管理系统。

软件缺陷管理系统(Defect Tracking System)适用于集中管理软件测试过程中发现缺陷的数据库程序,可以通过添加、修改、排序、查找、存储等操作来管理软件缺陷。利用软件缺陷跟踪管理系统便于查找和跟踪缺陷,因为对于大中型软件的测试过程而言,报告的缺陷总数可能会有成千上万个,如果没有缺陷跟踪管理系统的支持,要求查找某个错误,其难度和效率可想而知。

软件缺陷管理系统的作用如下。

1. 保持高效率的测试过程

由于软件缺陷跟踪管理系统一般都是通过测试小组内部局域网运行,因此打开和操作速度快。软件测试人员可随时向内部数据库添加新发现的软件缺陷,而且如果遗漏某项软件缺陷的内容,数据库系统将会及时给出提示,保证软件缺陷报告的完整性和一致性。软件缺陷验证工程师可以将主要的精力用在验证数据库中心报告的缺陷,这样就保证了效率。

2. 提高软件缺陷报告的质量

软件缺陷报告的一致性和正确性是衡量软件测试公司测试专业程度的指标之一。正确和完整地填写软件缺陷数据库的各项内容,可以保证测试工程师的缺陷报告格式统一。同时,引入软件缺陷跟踪管理系统可以从测试工具和测试流程上,保证不同测试技术背景的测试成员书写结构一致的软件缺陷报告。为了提高报告的效率,软件缺陷数据库的很多字段内容可以直接选择,而不必每次都手工来输入。

3. 实施实时管理,安全控制

软件缺陷查询、筛选、排序、添加、修改、保存、权限控制是数据库管理的基本功能和主要优势。通过方便的数据库查询和分类筛选,可迅速定位软件缺陷和统计软件缺陷的类型。通过权限设置,保证只有适当权限的人才能修改或删除软件缺陷,从而保证测试的质量。最后它还有利于跟踪和监控错误的处理过程和方法,可以方便地检查处理方法是否正确,跟踪处理者的姓名和处理时间,作为工作量统计和业绩考核的参考。

4. 有利于项目中成员间协同工作

软件缺陷跟踪管理系统可以作为测试人员、开发人员、项目负责人、缺陷评审人员协同工作的平台,便于他们及时掌握各种软件缺陷的当前状态,进而完成对应状态的测试工作。

目前已有的软件缺陷跟踪管理软件包括 Compuware 公司的 TrackRecord 软件(商业软件)、Mozilla 公司的 Buzilla 软件(免费软件),以及国内的微创公司的 BMS 软件,这些软件在功能上各有特点,可以根据实际情况选用。当然,也可以自己开发软件缺陷跟踪软件,例如基于 Notes 或是 ClearQuese 开发软件缺陷跟踪管理软件。

除此之外,作为一个软件缺陷跟踪管理系统,还必须注意权限分配的问题。软件缺陷记

录作为软件开发过程中的重要数据，不能轻易删除；对于已经关闭的软件缺陷，也不能随意进行修改。因此，软件缺陷跟踪管理系统必须设置严格的管理权限，非相关人员不得进行相应操作，修改相应数据。在这一点上，通过 Notes 也很容易控制。

6.6.6 手工报告和跟踪软件缺陷

显然，我们在测试过程过程中，对每一个软件缺陷都应该记录下来，如果使用上面所讲的软件缺陷跟踪管理系统，那么测试工具将自动记录软件缺陷的相关信息。如果测试采用手工记录和跟踪软件缺陷，那么有关软件缺陷的信息可以直接记录在相应的文档中。图 6-4 是根据 ANS/IEEE 829—1998 标准设计的软件缺陷报告文档。

公司名称：__________ Bug 报告：__________ Bug#：__________
软件：__________ 版本：__________
测试员：__________ 日期：__________
严重性：__________ 是否会重现： 是 否
标题：__________
描述：__________

解决方法：__________
解决日期：__________ 解决人：__________ 版本号：__________
解决描述：__________

重新测试人：__________ 测试版本号：__________ 测试日期：__________
重新测试描述：__________

签名：
策划：__________ 测试：__________
编程：__________ 项目管理：__________
销售：__________ 技术支持：__________

图 6-4 软件缺陷报告文档

6.7 软件测试的评测

为什么要进行软件测试的评测呢？软件测试的评测主要有两个目的：一是量化测试进程，判断软件测试进行的状态，以决定什么时候软件测试可以结束；二是为最后的测试或软件质量分析报告生成所需的量化数据，如缺陷清除率、测试覆盖率等。软件测试评测的结果是软件测试的一个阶段性结论，通过所生成的软件测试评测报告来确定软件测试是否完全

和成功。软件测试评测贯穿整个软件测试过程，可以在测试的每个阶段结束前进行，也可以在测试过程中的某一个时间进行。

测试的主要评测方法包括覆盖评测和质量评测。覆盖评测是对测试完全程度的评测，它建立在测试覆盖基础上，测试覆盖是由测试需求和测试用例的覆盖或已执行代码的覆盖表示的。质量评测是对测试对象（系统或测试的应用程序）的可靠性、稳定性以及性能的评测。质量评测建立在对测试结果的评估和对测试过程中确定的变更请求（缺陷）的分析基础上。

6.7.1 覆盖评测

覆盖测评指标用来度量软件测试的完全程度，所以可以将覆盖用做测试有效性的一个度量。最常用的覆盖测评是基于需求的测试覆盖和基于代码的测试覆盖，它们分别是指针对需求（基于需求的）或代码设计实施（基于代码的）而言的完全程度测评。

系统的测试活动应建立在一个测试覆盖策略基础上。如果需求已经完全分类，则基于需求的覆盖策略可能足以生成测试完全程度的可计量评测。例如，若已经确定了所有性能测试需求，则可以引用测试结果来得到测评，如已经核实了75%的性能测试需求等；如果应用基于代码的覆盖，则测试策略是根据测试已经执行的源代码的多少来表示。两种测试都可以手工计算得到，或通过测试自动化工具得到。

1. 基于需求的测试覆盖

基于需求的测试覆盖在测试生命周期中要评测多次，并在测试生命周期的里程碑处提供测试覆盖的标识（如已计划的、已实施的、已执行的和成功的测试覆盖）。测试覆盖率通过以下公式计算：

$$测试覆盖率 = T\hat{}(p,i,x,s)\ /\ \mathrm{RfT}$$

其中，T 是用测试过程或测试用例表示的测试（Test）数（已计划的、已实施的或成功的），RfT 是测试需求（Requirement for Test）的总数。

2. 基于代码的测试覆盖

基于代码的测试覆盖评测是测试过程中已经执行的代码的多少，与之相对的是要执行的剩余代码的多少。代码覆盖可以建立在控制流（语句、分支或路径）或数据流的基础上。基于代码的测试覆盖率通过以下公式计算：

$$测试覆盖率 = I\hat{}\mathrm{e}\ /\ \mathrm{TIic}$$

其中，I ^e 是用代码语句、代码分支、代码路径、数据状态判定点或数据元素名表示的已执行项目数，TIic（Total number of Items in the code）是代码中的项目总数。

6.7.2 质量评测

测试覆盖的测评提供了对测试完全程度的评价，而在测试过程中对已发现软件缺陷的测评提供了最佳的软件质量指标。因为质量是反映软件与需求相符程度的指标，所以在这种环境中，软件缺陷被标识为一种更改请求，在此更改请求中的测试对象是与需求不符的。

软件缺陷测评可能建立在各种方法上,这些方法种类繁多,从简单的软件缺陷计数到严格的统计建模不一而足。

严格的评估假定测试过程中软件缺陷达到的比率或发现的比率。常用模型假定该比率符合泊松分布,则有关软件缺陷率的实际数据可以适用于这一模型。生成的评估将评估当前软件的可靠性,并且预测继续测试并排除软件缺陷时可靠性如何增长。该评估被描述为软件可靠性增长建模,这是一个活跃的研究领域。由于该类型的评估缺乏工具支持,所以应该慎重平衡成本与其增加价值。

常用的测试有效性度量围绕软件缺陷分析来构造。软件缺陷分析就是分析软件缺陷在与软件缺陷相关联的一个或多个参数值上的分布。软件缺陷分析提供了一个软件可靠性指标,这些分析为揭示软件可靠性的缺陷趋势或缺陷分布提供了判断依据。

对于缺陷分析,常用的主要缺陷参数有 4 个。

- 状态:缺陷的当前状态(打开的、正在修复的和关闭的等)。
- 优先级:表示修复缺陷的重要程度和应该何时修复。
- 严重性:表示软件缺陷的恶劣程度,反映其对产品和用户的影响等。
- 起源:导致缺陷的原因及其位置,或排除该缺陷需要修复的构件。

缺陷分析通常以 4 类形式的度量提供缺陷评测。

- 缺陷发现率。
- 缺陷潜伏期。
- 缺陷密度。
- 整体软件缺陷清除率。

可以将缺陷计数作为时间的函数来报告,即创建缺陷趋势图或报告;也可以将缺陷计数作为一个或多个缺陷参数的函数来报告,如作为缺陷密度报告中采用的严重性或状态参数的函数。这些分析类型分别为揭示软件可靠性的缺陷趋势或缺陷分布提供了判断依据。

例如,预期缺陷发现率将随着测试进度和修复进度而最终减少。可以设定一个阈值,在缺陷发现率低于该阈值时才能部署软件。也可根据执行模型中的起源报告缺陷计数,以允许检测"较差的模块"、"热点"或需要再三修复的软件部分,从而指示一些更基本的设计缺陷。

这种分析中包含的缺陷必须是已确认的缺陷。不是所有报告的缺陷都是实际的缺陷,这是因为某些缺陷可能是扩展请求,超出了项目的规模,或描述的是已报告的缺陷。然而,需要查看并分析一下,为什么许多报告的缺陷不是重复的缺陷就是未经确认的缺陷,这样做是有价值的。

6.7.3 性能评测

评估测试对象的性能行为时,可以使用多种测评方法,测评侧重于获取与行为相关的数据,如响应时间、计时配置器、执行流、操作可靠性和限制。这些测评主要在评估测试活动中进行评估,但是也可以在执行测试活动中使用性能评测来评估测试进度和状态。

主要的性能测评如下。

- 动态监测:在测试执行过程中,实时获取并显示在执行的各测试用例的状态。

- 响应时间和吞吐量：测试对象针对特定测试用例的响应时间和吞吐量的评测。
- 百分比报告：数据已收集值的百分比计算与测评。
- 比较报告：代表不同测试执行情况的两个(或多个)数据集之间的差异或趋势。
- 追踪报告：测试用例和测试对象之间的消息和会话详细信息。

6.8 测试总结报告

测试总结报告的目的是总结测试活动的结果，并根据这些结果对测试进行评价。这种报告是测试人员对测试工作进行总结，并识别出软件的局限性和发生失效的可能性。在测试执行阶段的末期，应该为每个测试计划准备一份相应的测试总结报告。从本质上讲，测试总结报告是测试计划的扩展，起着对测试计划“封闭回路”的作用。

图 6-5 所示的是符合 IEEE 829—1998 软件测试文档编制标准的测试总结报告模板。

IEEE 829—1998 软件测试文档编制标准
测试总结报告模板

目录
1. 测试总结报告标识符
2. 概述
3. 差异
4. 综合评估
5. 结果总结
  5.1　已解决的意外事件
  5.2　未解决的意外事件
6. 评价
7. 建议
8. 活动总结
9. 审批

图 6-5　测试总结报告模板

1. 测试总结报告标识符

报告标识符是标识报告的唯一 ID，用来方便测试总结报告的管理、定位和引用。

2. 概述

这部分内容概要说明发生了哪些测试活动，包括软件的版本发布及环境等。这部分内容通常还包括测试计划、测试设计规格说明、测试规程和测试用例提供的参考信息等。

3. 差异

这部分内容主要描述计划的测试工作与真实发生的测试之间存在的所有差异。对于测试人员来说，这部分内容相当重要，因为它有助于测试人员掌握各种变更情况，并加深测试人员对今后如何改进测试计划过程的认识。

4. 综合评估

在这一部分中，应该对照在测试计划中规定的准则，对测试过程的全面性进行评价。这些准则是建立在测试清单、需求、设计、代码覆盖，或这些因素的综合结果基础之上的。在这里，需要指出那些覆盖不充分的特征或者特征集合，也可以对任何新出现的风险进行讨论。在这部分内容，还需要对所采用的测试有效性的所有度量进行报告和说明。

5. 结果总结

这部分内容用于总结测试结果。应该标识出所有已经解决的软件缺陷，并总结这些软件缺陷的解决方法；还要标识出所有未解决的软件缺陷。这部分内容还包括与缺陷及其分布相关的度量。

6. 评价

在这一部分中，应该对每个测试项，包括各个测试项的局限性进行总体评价。例如，对于可能存在的局限性，可以用这样一些语句来描述："系统不能同时支持 100 名以上的用户"，或者"如果吞吐量超出一定的范围，性能将会降至……"。这部分内容可能还会包括根据系统在测试期间所表现出的稳定性、可靠性或对测试期间观察到的失效的分析，以及对失效可能性进行的讨论等。

7. 建议

针对本次测试工作，提出自己的意见或建议。没有可填写"无"。

8. 活动总结

总结主要的测试活动和事件。总结资源消耗数据，如人员配置的总体水平、总的机器时间以及花在每一项主要测试活动上的时间等。这部分内容对于测试人员来说十分重要，因为这里记录的数据，可提供估计今后的测试工作所需的信息。

9. 审批

在这一部分，列出对这个报告享有审批权的所有人员的名字和职务。留出用于署名和填写日期的空间。理想情况下，我们希望审批这个报告的人员与审批相应的测试计划的人员相同，因为测试总结报告是对相应的计划所勾勒的所有活动的总结。通过签署这份文档，这些审批人员表明自己对报告中所陈述的结果持肯定态度，这份报告代表所有审批人的一致意见。如果有些评审人对这份报告的看法存有细微的分歧，他们也会签署这份文档，并可在文档中注明自己与他人存在的分歧意见。

6.9 本章小结

对软件项目的测试，除了应严格按照软件测试流程来实施之外，根据测试情况撰写测试报告和测试评测，对测试进行评估和质量分析，同样是一项关键的工作。应将发现的软件缺

陷及时地汇报出来，并对测试进行各方面的评测，最终撰写测试总结报告。

习题 6

1. 什么是软件缺陷？如何描述软件缺陷？
2. 软件缺陷有哪些类型？
3. 软件测试人员应该如何正确面对软件缺陷？
4. 什么是软件缺陷的生命周期？
5. 可采取哪些方法来分离和再现软件缺陷？
6. 如何进行软件测试的评测？



第7章 软件测试项目管理

软件测试项目的管理，一方面具有软件项目管理的共性，另一方面也具有软件测试自身的管理特点。软件测试项目管理是软件工程的保护性活动。它先于任何测试活动之前开始，且持续贯穿于整个测试项目的定义、计划和测试之中。

本章主要介绍软件测试项目管理知识，包括软件测试文档的编写、软件测试的组织与人员管理、软件测试过程管理、软件测试的配置管理、软件测试风险管理、软件测试的成本管理。

7.1 软件测试项目管理概述

测试项目是在一定的组织机构内，利用有限的人力和财力等资源，在指定的环境和要求下，对特定软件完成特定测试目标的阶段性任务。该任务应满足一定质量、数量和技术指标等要求。

测试项目一般具有如下一些基本特性：

(1) 项目的独特性；

(2) 项目的组织性；

(3) 测试项目的生命期；

(4) 测试项目的资源消耗特性；

(5) 测试项目目标冲突性；

(6) 测试项目结果的不确定因素。

测试项目管理就是以测试项目为管理对象，通过一个临时性的专门的测试组织，运用专门的软件测试知识、技能、工具和方法，对测试项目进行计划、组织、执行和控制，并在时间成本、软件测试质量等方面进行分析和管理活动(一种高级管理方法)。测试项目管理贯穿整个测试项目的生命周期，是对测试项目的全过程进行管理。

测试项目管理有以下基本特征：

(1) 系统工程的思想贯穿测试项目管理的全过程；

(2) 测试项目管理的组织有一定的特殊性；

(3) 测试项目管理的要点是创造和保持一个使测试工作顺利进行的环境，使置身于这个环境中的人员能在集体中协调工作以完成预定的目标；

(4) 测试项目管理的方法、工具和技术手段具有先进性。

测试项目范围管理就是界定项目所必须包含且只需包含的全部工作，并对其他的测试

项目管理工作起指导作用,以确保测试工作顺利完成。

项目目标确定后,下一步过程就是确定需要执行哪些工作或者活动来完成项目的目标,即要确定一个包含项目所有活动在内的一览表。准备这样的一览表通常有两种方法:一种是让测试小组利用“头脑风暴法”根据经验,集思广益来形成。这种方法比较适合小型测试项目。另一种是对更大更复杂的项目建立一个工作分解结构(Work Breakdown Structure,WBS)和任务的一览表。工作分解结构是将一个软件测试项目分解成易于管理的更多部分或细目,所有这些细目构成了整个软件测试项目的工作范围。工作分解结构是进行范围规划时所使用的重要工具和技术之一,它是测试项目团队在项目期间要完成或生产出的最终细目的等级树,组织并定义了整个测试项目的范围,未列入工作分解结构的工作将排除在项目范围之外。

进行工作分解是非常重要的工作,它在很大程度上决定项目能否成功。对于细分的所有项目要素需要统一编码,并按规范化进行要求。这样,WBS的应用将给所有的项目管理人员提供一个一致的基准,即使项目人员变动时,也有一个可以互相理解和交流沟通的平台。

7.2　软件测试文档

测试文档是对要执行的软件测试及测试的结果进行描述、定义、规定和报告的任何书面或图示信息。由于软件测试是一个很复杂的过程,同时也涉及软件开发中其他一些阶段的工作,因此,必须把对软件测试的要求、规划、测试过程等有关信息和测试的结果,以及对测试结果的分析、评价,以正式的文档形式给出。

测试文档对于测试阶段工作的指导与评价作用更是非常明显的。需要特别指出的是,在已开发的软件投入运行的维护阶段,常常还要进行再测试或回归测试,这时还会用到测试文档。因此,测试文档的编写是测试管理的一个重要组成部分。

7.2.1　测试文档的作用

测试文档的重要作用可从以下几个方面看出:

(1) 促进项目组成员之间的交流沟通;

(2) 便于对测试项目的管理;

(3) 决定测试的有效性;

(4) 检验测试资源;

(5) 明确任务的风险;

(6) 评价测试结果;

(7) 方便再测试;

(8) 验证需求的正确性。

7.2.2　测试文档的类型

根据测试文档所起的不同作用,通常把它分成两类,即前置作业文档和后置作业文档。测试计划及测试用例的文档属于前置作业文档。后置作业文档是在测试完成后提交的,主要包括软件缺陷报告和分析总结报告。

7.2.3 主要软件测试文档

1. 软件测试文档模板

图 7-1 给出了 IEEE 829—1998 软件测试文档标准(更新版本为 IEEE 829—2008)中定义的软件测试主要文档的类型。

IEEE 829－1998 软件测试文档编制标准 软件测试文档模板
目录
测试计划
测试设计规格说明
测试用例说明
测试规程规格说明
测试日志
测试缺陷报告
测试总结报告

图 7-1 软件测试文档模板

2. 软件测试计划

测试计划文档主要对软件测试项目、所需要进行的测试工作、测试人员所应该负责的测试工作、测试过程、测试所需的时间和资源,以及测试风险等做出预先的计划和安排,其模板如图 7-2 所示。

IEEE 829－1998 软件测试文档编制标准 软件测试计划文档模板
目录
1. 测试计划标识符
2. 介绍
3. 测试项
4. 需要测试的功能
5. 方法(策略)
6. 不需要测试的功能
7. 测试项通过/失败的标准
8. 测试中断和恢复的规定
9. 测试完成所提交的材料
10. 测试任务
11. 环境需求
12. 职责
13. 人员安排与培训需求
14. 进度表
15. 潜在的问题和风险
16. 审批

图 7-2 软件测试计划文档模板

3. 测试设计规格说明

用于每个测试等级,以指定测试集的体系结构和覆盖跟踪,其模板如图 7-3 所示。

IEEE 829—1998 软件测试文档编制标准 软件测试设计规格说明文档模板
目录
测试设计规格说明标识符
待测试特征
方法细化
测试标识
通过/失败准则

图 7-3 软件测试设计规格说明文档模板

4. 软件测试用例规格说明文档

用于描述测试用例,其模板如图 7-4 所示。

IEEE 829—1998 软件测试文档编制标准 软件测试用例规格说明文档模板
目录
测试用例规格说明标识符
测试项
输入规格说明
输出规格说明
环境要求
特殊规程需求
用例之间的相关性

图 7-4 软件测试用例规格说明文档模板

5. 测试规程

用于指定执行一个测试用例集的步骤。

6. 测试日志

由于记录测试的执行情况不同,可根据需要选用。

7. 软件缺陷报告

用来描述出现在测试过程或软件中的异常情况,这些异常情况可能存在于需求、设计、代码、文档或测试用例中。

8. 测试总结报告

用于报告某个测试完成情况。

7.3 软件测试的组织与人员管理

从软件的生存周期看，测试往往指对程序的测试，这样做的优点是被测对象明确，测试的可操作性相对较强。但是，由于测试的依据是规格说明书、文档和使用说明书，如果设计有错误，测试的质量就难以保证。即使测试后发现是设计的错误，这时，修改的代价是相当昂贵的。因此，较理想的做法应该是对软件的开发过程，按软件工程各阶段形成的结果，分别进行严格的审查。

为了确保软件的质量，对各个过程应进行严格的管理。虽然测试是在实现且验证后进行的，实际上，测试的准备工作在分析和设计阶段就开始了。

7.3.1 测试的过程及组织

当设计工作完成以后，就应该着手测试的准备工作了。一般来讲，可由一位对整个系统设计熟悉的设计人员编写测试大纲，明确测试的内容和测试通过的准则，设计完整合理的测试用例，以便系统实现后进行全面测试。

实现组将所开发的程序验证后，提交测试组，即可由测试负责人组织测试。测试一般可按下列方式和步骤组织。

(1) 测试人员要仔细阅读有关资料，包括规格说明、设计文档、使用说明书及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试用例，做好测试前的准备工作。

(2) 为了保证测试的质量，将测试过程分成几个阶段，即：代码审查、单元测试、集成测试和验收测试。

(3) 代码会审。代码会审是由一组人通过阅读、讨论和争议对程序进行静态分析的过程。会审小组由组长、2～3 名程序设计和测试人员及程序员组成。会审小组在充分阅读待审程序文本、控制流程图及有关要求、规范等文件基础上，召开代码会审会，程序员逐句讲解程序的逻辑，并展开热烈的讨论甚至争议，以揭示错误的关键所在。实践表明，程序员在讲解过程中能发现许多自己原来没有发现的错误，而讨论和争议则进一步促使了问题的暴露。例如，对某个局部性小问题修改方法的讨论，可能发现与之有牵连的甚至能涉及到模块的功能说明、模块间接口和系统总结构的大问题，导致对需求定义的重定义、重设计验证，从而大大改善软件的质量。

(4) 单元测试。单元测试集中在检查软件设计的最小单位——模块上，通过测试发现实现该模块的实际功能与定义该模块的功能说明不符合的情况，以及编码的错误。由于模块规模小、功能单一、逻辑简单，测试人员有可能通过模块说明书和源程序，清楚地了解该模块的 I/O 条件和模块的逻辑结构。通常采用结构测试(白盒法)的用例，以求尽可能达到彻底测试，然后辅以功能测试(黑盒法)的用例，使之对任何合理和不合理的输入都能鉴别和响应。高可靠性的模块是组成可靠系统的坚实基础。

(5) 集成测试。集成测试是将模块按照设计要求组装起来同时进行测试，主要目标是发现与接口有关的问题。如数据穿过接口时可能丢失；一个模块与另一个模块可能有由于

疏忽的问题而造成有害影响；把子功能组合起来可能不产生预期的主功能；个别看起来是可以接受的误差可能积累到不能接受的程度；全程数据结构可能有错误等。

(6) 验收测试。验收测试的目的是向未来的用户表明系统能够像预定要求那样工作。经集成测试后，已经按照设计把所有的模块组装成一个完整的软件系统，接口错误也已经基本排除了，接着就应该进一步验证软件的有效性，这就是验收测试的任务，即保证软件的功能和性能如同用户所合理期待的那样。

经过上述的测试过程，软件基本满足开发的要求，测试宣告结束，经验收后，将软件提交给用户。

7.3.2　测试方法的应用

集成测试及其后的测试阶段，一般采用黑盒方法。其策略包括：

(1) 用边值分析法和(或)等价分类法提出基本的测试用例；

(2) 用猜测法补充新的测试用例；

(3) 如果在程序的功能说明中含有输入条件的组合，宜在一开始就用因果图法，然后再按以上(1)、(2)两步进行。

单元测试的设计策略稍有不同。因为在为模块设计程序用例时，可以直接参考模块的源程序。所以单元测试的策略，总是把白盒法和黑盒法结合运用。具体做法有两种。

(1) 先仿照上述步骤用黑盒法提出一组基本的测试用例，然后用白盒法作验证。如果发现用黑盒法产生的测试用例未能满足所需的覆盖标准，就用白盒法增补新的测试用例来满足它们。覆盖的标准应该根据模块的具体情况确定。对可靠性要求较高的模块，通常要满足条件组合覆盖或路径覆盖标准。

(2) 先用白盒法分析模块的逻辑结构，提出一批测试用例，然后根据模块的功能用黑盒法进行补充。

7.3.3　测试的人员组织

为了保证软件的开发质量，软件测试应贯穿于软件定义与开发的整个过程。因此，对分析、设计和实现等各阶段所得到的结果，包括需求规格说明、设计规格说明及源程序都应进行软件测试。基于此，测试人员的组织也应是分阶段的。

1. 需求规格说明评审

软件的设计和实现都是基于需求分析规格说明进行的。需求分析规格说明是否完整、正确、清晰是软件开发成败的关键。为了保证需求定义的质量，应对其进行严格的审查。审查小组由下列人员组成：

- 组长 1 人；
- 成员包括系统分析员，软件开发管理者，软件设计、开发和测试人员和用户。

2. 设计评审

软件设计是将软件需求转换成软件表示的过程。主要描绘出系统结构、详细的处理过

程和数据库模式。

在设计评审阶段应按照需求的规格说明对系统结构的合理性、处理过程的正确性进行评价，同时利用关系数据库的规范化理论对数据库模式进行审查。评审小组由下列人员组成：

- 组长 1 人；
- 成员包括系统分析员、软件设计人员、测试负责人员各一人。

3. 程序的测试

软件测试是整个软件开发过程中交付用户使用前的最后阶段，是软件质量保证的关键。软件测试在软件生存周期中横跨两个阶段。通常在编写出每一个模块之后，就对它进行必要的测试(单元测试)。编码与单元测试属于软件生存周期中的同一阶段。该阶段的测试工作，由编程组内部人员进行交叉测试(避免编程人员测试自己的程序)。这一阶段结束后，进入软件生存周期的测试阶段，对软件系统进行各种综合测试。测试工作由专门的测试组完成，测试小组由下列人员组成：

- 组长 1 人，负责整个测试的计划、组织工作；
- 成员 3～5 人，由具有一定的分析、设计和编程经验的专业人员组成。

7.3.4 软件测试文件

软件测试文件描述要执行的软件测试及测试的结果。由于软件测试是一个很复杂的过程，同时也涉及软件开发其他一些阶段的工作，对于保证软件的质量和它的运行有着重要意义，必须把对它们的要求、过程及测试结果以正式的文件形式写出。测试文件的编写是测试工作规范化的一个组成部分。

测试文件不只在测试阶段才考虑，在软件开发的需求分析阶段就应开始着手，因为测试文件与用户有着密切的关系。在设计阶段的一些设计方案也应在测试文件中得到反映，以利于设计的检验。测试文件对于测试阶段工作的指导与评价作用更是非常明显的。需要特别指出的是，在已开发的软件投入运行的维护阶段，常常还要进行再测试或回归测试，这时仍需用到测试文件。

1. 测试文件的类型

根据测试文件所起的作用不同，通常把测试文件分成两类，即测试计划和测试分析报告。测试计划详细规定测试的要求，包括测试的目的和内容、方法和步骤，以及测试的准则等。由于要测试的内容可能涉及到软件的需求和软件的设计，因此必须及早开始测试计划的编写工作。不应在着手测试时，才开始考虑测试计划。通常，测试计划的编写从需求分析阶段开始，到软件设计阶段结束时完成。测试报告用来对测试结果分析说明，经过测试后，证实了软件具有的能力，以及它的缺陷和限制，测试报告给出评价的结论性意见，这些意见既是对软件质量的评价，又是决定该软件能否交付用户使用的依据。由于测试报告要反映测试工作的情况，自然要在测试阶段内编写。

2. 测试文件的使用

测试文件的重要性表现在以下几个方面。

（1）验证需求的正确性：测试文件中规定了用以验证软件需求的测试条件，研究这些测试条件对弄清用户需求的意图是十分有益的。

（2）检验测试资源：测试计划不仅要用文件的形式把测试过程规定下来，还应说明测试工作必不可少的资源，进而检验这些资源是否可以得到，即它的可用性如何。如果某个测试计划已经编写出来，但所需资源仍未落实，那就必须及早解决。

（3）明确任务的风险：有了测试计划，就可以弄清楚测试可以做什么，不能做什么。了解测试任务的风险有助于对潜伏的可能出现的问题事先做好思想上和物质上的准备。

（4）生成测试用例：测试用例的好坏决定着测试工作的效率，选择合适的测试用例是做好测试工作的关键。在测试文件编制过程中，按规定的要求精心设计测试用例有重要的意义。

（5）评价测试结果：测试文件包括测试用例，即若干测试数据及对应的预期测试结果。完成测试后，将测试结果与预期的结果进行比较，便可对已进行的测试提出评价意见。

（6）再测试：测试文件规定的和说明的内容对维护阶段由于各种原因的需求进行再测试时，是非常有用的。

（7）决定测试的有效性：完成测试后，把测试结果写入文件，这对分析测试的有效性，甚至整个软件的可用性提供了依据。同时还可以证实有关方面的结论。

3. 测试文件的编制

在软件的需求分析阶段，就开始测试文件的编制工作，各种测试文件的编写应按一定的格式进行。

7.4　软件测试过程管理

7.4.1　软件测试过程模型

1. 软件测试过程模型介绍

1）V 模型

V 模型最早是由 Paul Rook 在 20 世纪 80 年代后期提出的，旨在改进软件开发的效率和效果。V 模型反映出了测试活动与分析设计活动的关系，如图 7-5 所示。图 7-5 描述了基本的开发过程和测试行为，非常明确地标注了测试过程中存在的不同类型的测试，以及这些测试阶段和开发过程期间各阶段的对应关系。

V 模型指出，单元测试和集成测试应检测程序执行是否满足软件设计的要求；系统测试应检测系统功能、性能的质量特性是否达到系统要求的指标；验收测试应确定软件的实现是否满足用户需要或合同的要求。但 V 模型存在一定的局限性，它仅仅把测试作为在编码之后的一个阶段，是针对程序进行的寻找错误的活动，而忽视了测试活动对需求分析、系统设计等活动的验证和确认的功能。

2）W 模型

W 模型由 Evolutlf 公司提出，相对于 V 模型，W 模型增加了软件各开发阶段中应同步

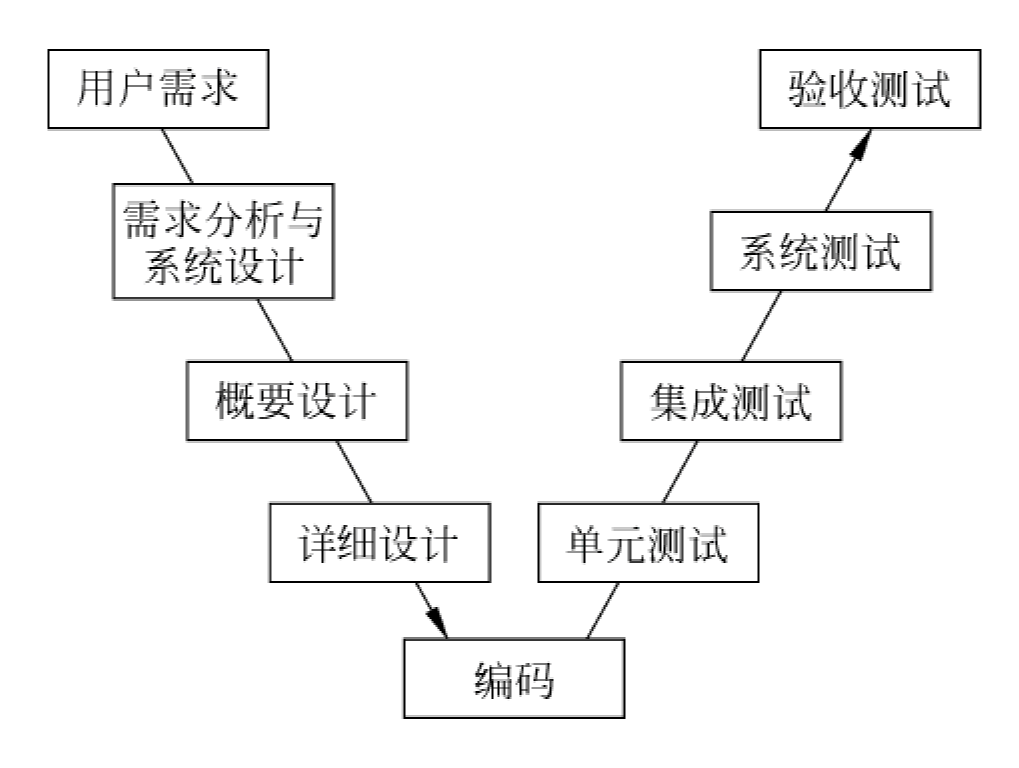


图 7-5 软件测试 V 模型

进行的验证和确认活动，如图 7-6 所示。W 模型由两个 V 字形模型组成，分别代表测试与开发过程，明确表示出了测试与开发的并行关系。

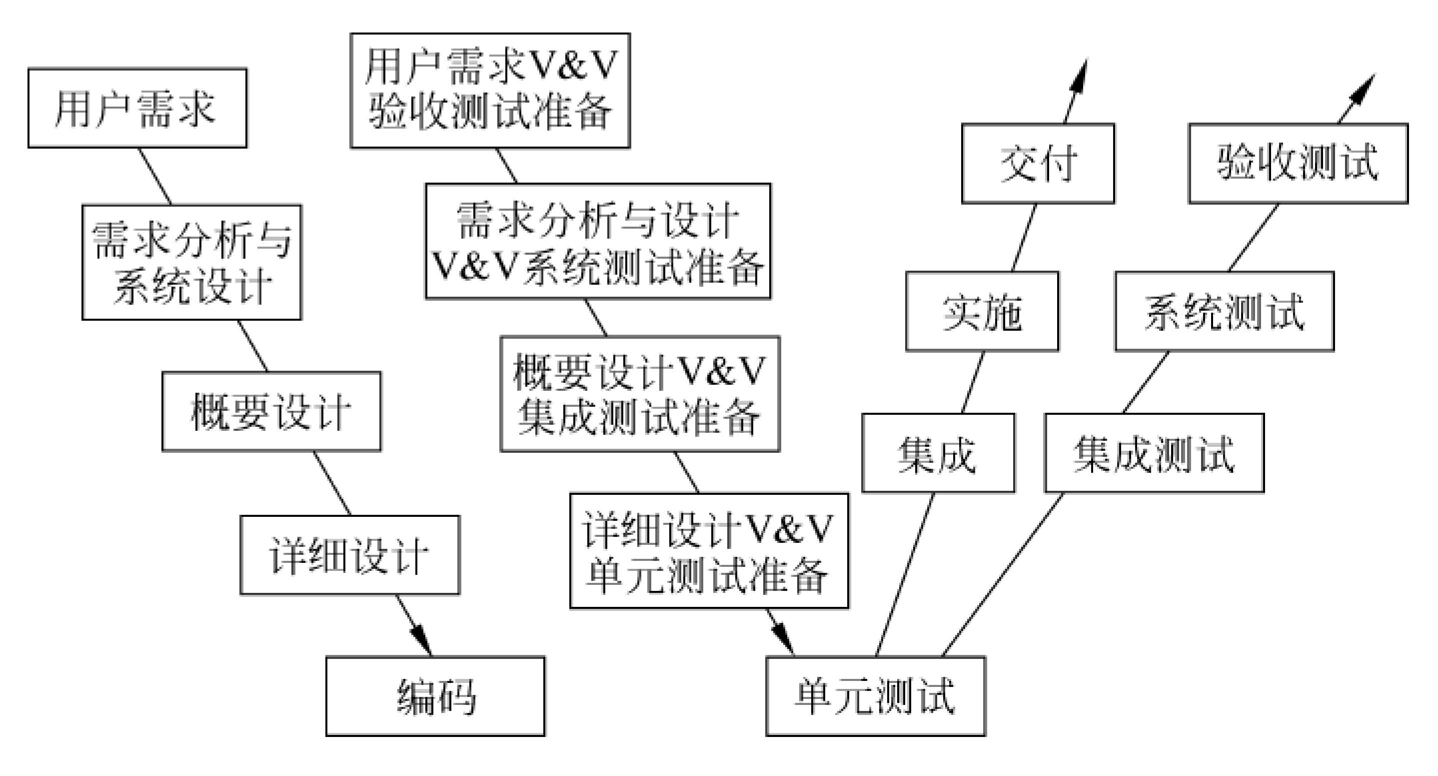


图 7-6 软件测试 W 模型

W 模型强调测试伴随整个软件开发周期，而且测试的对象不仅仅是程序，需求、设计等同样需要测试，也就是说，测试与开发是同步进行的。W 模型有利于尽早、全面地发现问题。例如，需求分析完成后，测试人员就应该参与对需求的验证和确认活动，尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，这将显著减少总体测试时间，加快项目进度。

但 W 模型也存在局限性。在 W 模型中，需求、设计、编码等活动被视为串行的，同时，测试和开发活动也保持着一种线性的前后关系，上一阶段结束后才可正式开始下一个阶段工作。这样就无法支持迭代的开发模型。对于当前软件开发复杂多变的情况，W 模型并不能解除测试管理面临的困惑。

3）H 模型

V 模型和 W 模型均存在一些不妥之处。如前所述，它们都把软件的开发视为需求、设计、编码等一系列串行的活动，而事实上，这些活动在大部分时间内是可以交叉进行的，所以，相应的测试之间也不存在严格的次序关系。同时，各层次的测试（单元测试、集成测试、系统测试等）也存在反复触发、迭代的关系。

为了解决以上问题，有专家提出了 H 模型。它将测试活动完全独立出来，形成了一个

完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来，如图 7-7 所示。

图 7-7 中标注的“其他流程”可以是任意的开发流程。例如，设计流程或编码流程。H 模型揭示了一个原理：软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行。H 模型指出软件测试要尽早准备，尽早执行。不同的测试活动可以按照某个次序先后进行，也可能是反复的，只要某个测试达到准备关键点，测试执行活动就可以开展。

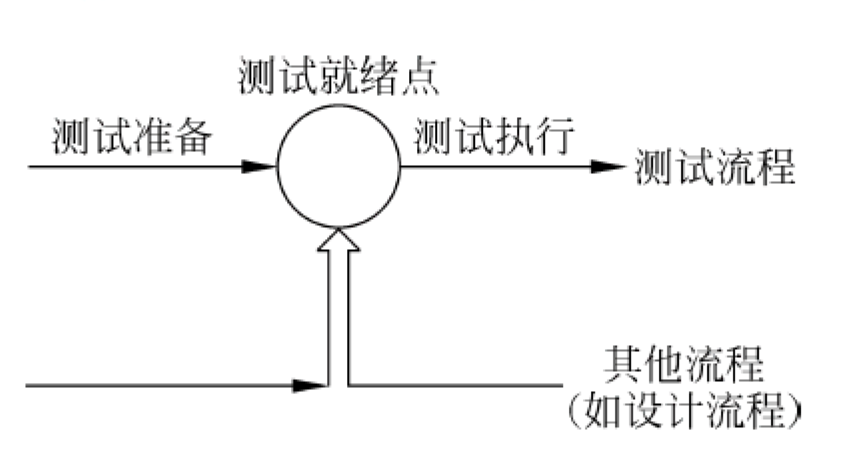


图 7-7　软件测试 H 模型

(4) 其他模型

除上述几种常见模型外，业界还流传着其他几种模型，例如前置测试模型、X 模型等。前置测试模型体现了开发与测试的结合，要求对每一个交付内容进行测试。X 模型提出针对单独的程序片段进行相互分离的编码和测试，此后通过频繁的交接，经集成最终合成为可执行的程序。这些模型都针对其他模型的缺点提出了一些修正意见，但本身也可能存在一些不周到的地方。所以在测试过程管理中，正确选取过程模型或者根据自身的实际情况制定一个模型是一个非常关键问题。

2. 软件测试过程模型的选取策略

前面介绍的测试过程模型中列出了各种模型的优缺点。测试人员在实际测试过程中应该尽可能地应用各模型中对项目有实用价值的方面，不能强行地为使用模型而使用模型。在测试实践中，大多采用的方法是：以 W 模型作为框架，及早、全面地开展测试。同时灵活运用 H 模型独立测试的思想，在达到恰当的关键点时就应该开展独立的测试工作，并将测试工作进行迭代，最终完成测试目标。

7.4.2　软件测试过程管理

软件测试过程管理如图 7-8 所示。

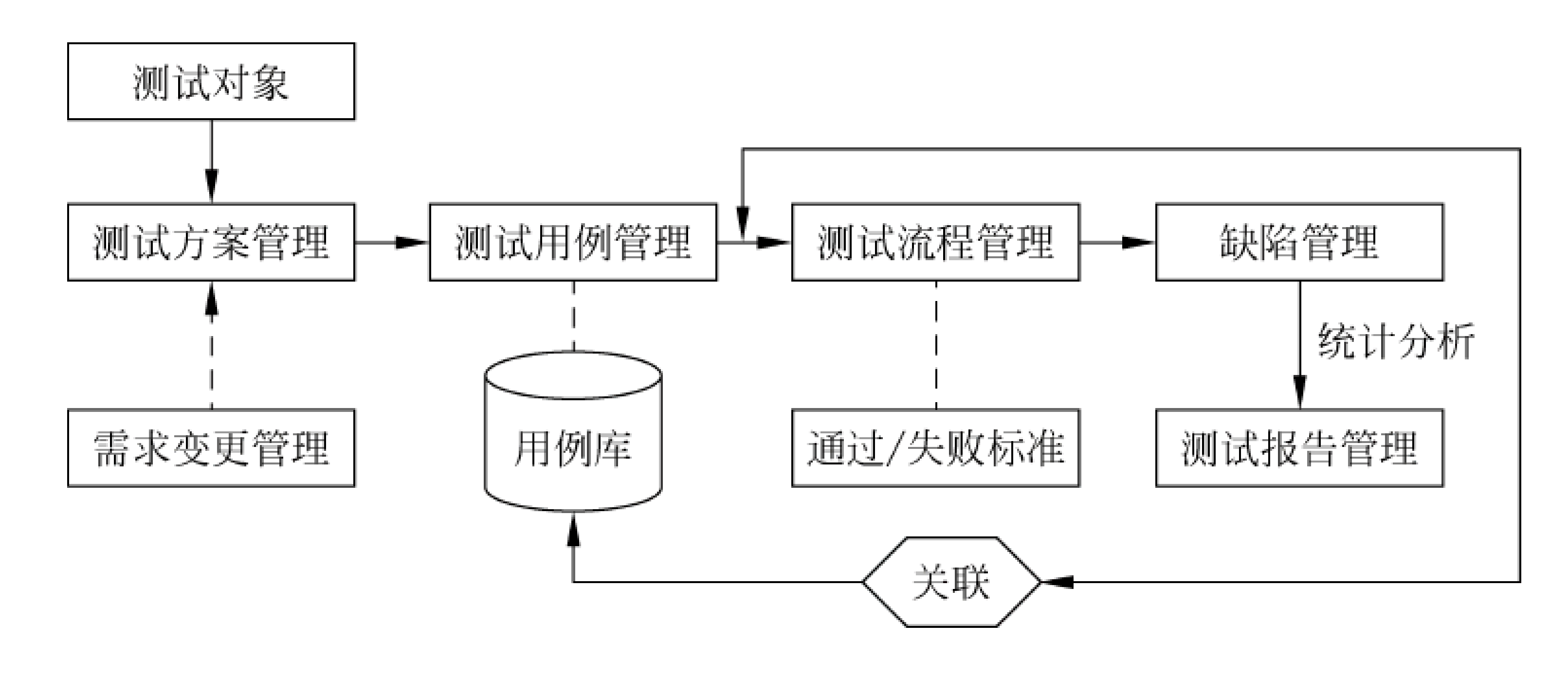


图 7-8　软件测试过程管理

应根据测试需求、测试计划，对测试过程中每个状态进行记录、跟踪和管理，并提供相关的分析和统计功能，生成和打印各种分析统计报表。通过对详细记录的分析，形成较为完整的软件测试管理文档，避免同样的错误在软件开发过程中再次发生，从而提高软件开发质量。

7.4.3 软件测试过程管理理念

生命周期模型为我们提供了软件测试的流程和方法，为测试过程管理提供了依据。但实际的测试工作是复杂而烦琐的，可能不会有哪种模型完全适用于某项测试工作。所以，我们应该从不同的模型中抽象出符合实际现状的测试过程管理理念，依据这些理念来策划测试过程，以不变应万变。当然测试管理牵涉的范围非常广泛，包括过程定义、人力资源管理、风险管理等，下面介绍几种管理理念。

1. 尽早测试

"尽早测试"是从 W 模型中抽象出来的理念。测试并不是在代码编写完成之后才开展的工作，测试与开发是两个相互依存的并行的过程，测试活动在开发活动的前期已经开展。

"尽早测试"有两方面的含义：第一，测试人员早期就参与软件项目，及时开展测试的准备工作，包括编写测试计划、制定测试方案以及准备测试用例；第二，尽早地开展测试执行工作，一旦代码模块完成就应该及时开展单元测试，一旦代码模块被集成为相对独立的子系统，便可以开展集成测试，一旦有 Build(版本)提交，便可以开展系统测试工作。

由于及早地开展了测试准备工作，测试人员能够较早地了解测试的难度，预测测试风险，从而有效提高测试效率、规避测试风险等，大大降低 Bug 修复成本。但是需要注意，"尽早测试"并非盲目提前测试活动，测试活动开展的前提是达到必需的测试关键点。

2. 全面测试

软件是程序、数据和文档的集合，那么对软件进行测试，就不仅仅是对程序的测试，还应包括软件"附属产品"的"全面测试"，这是 W 模型中一个重要的思想。需求文档、设计文档作为软件的阶段性产品，直接影响到软件的质量。阶段产品质量是软件质量的量的积累，不能把握这些阶段产品的质量将导致最终软件质量的不可控。

"全面测试"也包含两层含义：第一，对软件的所有产品进行全面的测试，包括需求、设计文档、代码、用户文档以及测试进度和测试策略的调整、需求变更等；第二，软件开发及测试人员(有时包括用户)全面地参与到测试工作中，并且对测试过程进行全面跟踪，例如对需求的验证和确认活动，就需要开发、测试及用户的全面参与，毕竟测试活动并不仅仅是为了保证软件运行正确，同时还要保证软件用户的需求。建立完善的度量和分析机制，通过对测试过程的度量，及时了解测试过程信息，以便调整测试策略等。

3. 独立的、迭代的测试

软件开发瀑布模型只是一种理想状况。为适应不同的需要，人们在软件开发过程中摸索出了如螺旋、迭代等诸多模型，若需求、设计、编码工作是重叠并反复进行，这时的测试工作也是迭代和反复的。如果不能将测试从开发中抽象出来进行管理，势必使测试管理陷入困境。

软件测试与软件开发是紧密结合的，但并不代表测试是依附于开发的一个过程，测试活动是独立的。"独立的、迭代的测试"着重强调了测试的关键点，也就是说，只要测试条件成

熟,测试准备活动完成,测试的执行活动就可以开展。

因此,测试人员在遵循尽早测试、全面测试等测试过程管理理念的同时,应当将测试过程从开发过程中适当地抽象出来,作为一个独立的测试过程进行管理。时刻把握独立的、迭代测试的理念,减小因开发模型的繁杂给测试管理工作带来的不便。对于软件过程中不同阶段的产品和不同的测试类型,只要测试准备工作就绪,就可以及时开展测试工作,把握产品质量。

7.4.4　软件测试过程管理实践

下面以一个实际项目系统测试过程的几个关键过程管理行为为例,阐述上节中提出的测试理念。

在一个办公自动化(OA)系统的管理项目中,由于前期需求不明确,开发周期相对较长,为了对项目进行更好的跟踪和管理,项目采用增量和迭代模型进行开发。整个项目开发共分三个阶段完成:第一阶段,实现进销存的简单的功能和工作流;第二阶段,实现固定资产管理、财务管理,并完善第一阶段的进销存功能;第三阶段,增加办公自动化的管理(OA)。该项目每一阶段工作是对上一阶段成果的一次迭代完善,同时将新功能进行了一次叠加。

1. 把握需求

在本系统开发过程中,需求的获取和完善贯穿每个阶段。对需求的把握很大程度上决定了软件测试是否能够成功。系统测试不仅仅确认软件是否正确实现功能,同时还要确认软件是否满足用户的需要。依据“尽早测试”和“全面测试”原则,在需求的获取阶段,测试人员参与到了对需求的讨论之中。测试人员与开发人员及用户一起讨论需求的完善性与正确性,同时从可测试性角度为需求文档提出建议。这些建议对开发人员来说,是从一个全新的思维角度提出的约束。同时,测试人员结合前期对项目的把握,很容易制定出了完善的测试计划和方案,对各阶段产品的测试方法及进度、人员安排进行了策划,使整个项目的进展有条不紊。

2. 变更控制

变更控制体现的是“全面测试”理念。在软件开发过程中,变更往往是不可避免的,变更也是造成软件风险的重要因素。在本系统测试中,仅第一阶段就发生了多次需求变更,调整了多次进度计划。依据“全面测试”理念,测试人员密切关注开发过程,跟随进度计划,变更调整测试策略,依据需求变更及时补充和完善测试用例。由于充分的测试准备工作,在测试执行过程中,没有废弃一个测试用例,测试的进度并没有因为变更而受到过多影响。

3. 度量与分析

对测试过程的度量与分析同样体现了“全面测试”理念。对测试过程的度量有利于及时把握项目情况,对过程数据进行分析,很容易发现优缺点,找出需要改进的地方,及时调整测试策略。

在此 OA 项目中，测试人员在测试过程中对不同阶段的 Bug 数量进行了度量，并分析测试执行是否充分，如图 7-9 所示。通过分析我们得出：相同时间间隔内发现的 Bug 数量呈收敛状态，测试是充分的。在 Bug 数量收敛的状态下结束细测是恰当的。

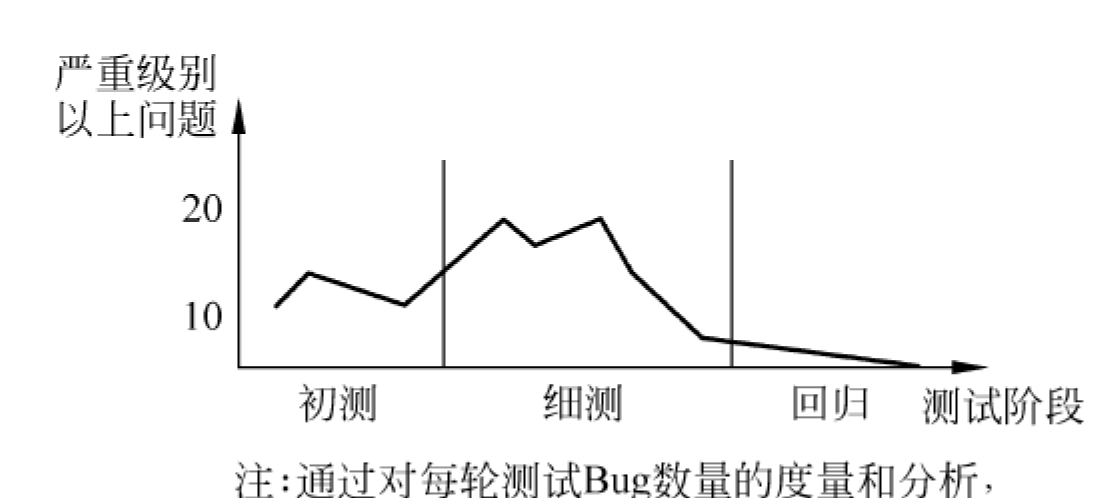


图 7-9　对测试过程的 Bug 数量进行度量，显示测试是充分的

测试中，我们的测试人员对不同功能点的测试数据覆盖率和发现问题数进行度量，以便分析测试用例的充分度与 Bug 发现率之间的关系。如表 7-1 所示，对类似模块进行对比发现：某一功能点上所覆盖的测试数据组越多，Bug 的用例发现率越高。如果再结合工作量、用例执行时间等因素进行统计分析，便可以找到适合实际情况的测试用例书写粒度，从而帮助测试人员判断测试成本和收益间的最佳平衡点。

表 7-1　测试数据覆盖率与 Bug 发现率对应表

模块名称	功能点数（个）	测试数据数（组）	测试数据覆盖率（组/每功能点）	Bug 的用例发现率（%）
模块 AA	6	75	12.5	40(6/15)
模块 BB	30	96	3.3	17(7/42)
模块 CC	15	87	5.1	18(5/28)
模块 DD	16	46	2.8	23(5/22)
…	…	…	…	…

说明：通过统计可以得出测试数据与 Bug 发现率之间的关系，便于及时调整测试用例编写策略。

所有这些度量都是对测试全过程进行跟踪的结果，是及时调整测试策略的依据。对测试过程的度量与分析有效地提高了测试效率，降低了测试风险。同时，度量与分析也是软件测试过程可持续改进的基础。测试人员在整个测试过程中，也加入一些“独立的、迭代的测试”的理念。

7.4.5　软件测试过程可持续改进

软件测试是一项复杂、具有创造性的工作，虽然已有了一些方法、理念，但都不是很完善的，许多问题还有待进一步研究和探索，使用时仍然需要测试人员的经验和创造力，这就注定测试过程也需要不断的改进。通过掌握各个组织的最新测试方法、理念，可使测试过程中发现的错误能够及时得到解决，修正产品，使应用系统更加完善，产品的质量更高。掌握了软件测试过程的可持续过程改进方法，测试过程管理将得到不断完善，测试活动也将始终处于优化状态中。

7.5　软件测试的配置管理

7.5.1　进行测试配置管理的必要性

在软件测试中缺少了测试的配置管理我们是做不好测试工作的。我们进行软件测试就是为了以最少的时间、最少的人力物力去尽可能多地发现软件中潜藏着的各种错误和缺陷。但是伴随着软件测试工作量的加大，软件企业仅仅投入更多的人力物力以及其他各种资源是不够的，还需要思考除此之外怎样才能更好地进行软件测试。为此，我们应该对测试人员、测试环境、生产环境进行配置管理。只有建立了完整的合理的软件测试配置管理体系，软件测试工作才能更好地进行，更加完美地完成测试目标。

如果在软件测试过程中缺少了测试配置管理将会造成极其严重的后果。据我们实际调查了解到，在日常的软件研发工作中，很多软件企业都会或多或少在软件测试时遇到问题，而这些问题的产生都是因为在测试过程中缺乏配置管理流程和工具。因为人员具有频繁的流动性，并且在组织的过程中会产生知识和财富的流失，再加上现代社会的激烈竞争，如果一个企业没有设计配置管理流程和使用必要的配置管理工具，就可能会因此而造成无可估量的损失，甚至导致整个软件项目的崩溃。因而作为一个软件企业，必须要做到及时了解项目的进展状况，对项目进行管理，遇到突发状况及早解决。软件工程思想发展到现在，认为在软件过程中如果能够越早发现缺陷和风险，这时只要采取相应的措施，所要付出的代价就越小。缺乏配置管理流程的一个很明显之处就是测试过程中缺乏并发执行的手段，没有了配置管理的支持，软件过程中的并发执行将会变得十分困难。这时往往会造成修改过的Bug重复出现，又或者几个人员进行相同的测试工作和进程，从而产生不必要的浪费。如果企业不能很清晰很流畅地对整个软件测试过程进行管理，就会造成测试工作的不同步、不一致。在测试工作中需要测试人员完成的没有完成，而暂时不需要或者以后要求再完成的却首先完成了，这样会增加测试工作的复杂性和难度，因此我们需要在软件测试中进行配置管理。

7.5.2　测试配置管理的方法和内容

既然测试配置管理在软件测试中如此重要，企业该如何进行测试配置管理呢。我们首先简单谈谈软件测试的测试配置管理体系。它一般由两种方法构成：那就是应用过程方法和系统方法。意思就是在测试过程中，我们应该把测试管理这块单独作为一个系统去对待。识别并且管理组成这个系统的每个过程，从而实现在测试工作开始之初设定的目标。在上面的基础之上，我们还要做到使这些过程在测试工作中能够协同作用，互相促进，最终使它们的总体作用更大。软件测试配置管理的主要目标是在设定的条件限制下，应当尽最大的努力去发现和排除软件缺陷。测试配置管理其实是包含在软件配置管理中的，是软件配置管理的子集。测试配置管理作用于软件测试的各个阶段，贯穿于整个测试过程之中。它的管理对象包括以下内容：测试方案、测试计划或者测试用例、测试工具、测试版本、测试环境以及测试结果等。这些就构成了软件测试配置管理的全部内容。

1. 测试配置管理的目标和阶段

现在我们了解软件测试配置管理的目标：第一步是在测试过程中控制和审计测试活动的变更，第二步是在测试过程中随着测试项目的里程碑，同步建立相应的基线；第三步是记录并且跟踪测试活动过程中的变更请求；第四步是在测试过程中针对相应的软件测试活动或者产品，测试人员应将它们标识为控制并且是可用的。

软件测试配置管理的阶段：第一阶段为需求阶段，我们要进行客户需求调研和软件需求分析；第二阶段为设计阶段，在这个阶段我们要进行概要设计和详细设计工作；第三阶段为编码阶段，这时我们主要进行的工作是编码；第四个阶段是测试和试运行阶段，在这个阶段我们要进行单元测试、用户手册编写、集成测试、系统测试、安装培训、试运行和安装运行这些工作；第五阶段也就是最后一个阶段，是正式运行及维护阶段，这时要做的是对产品进行发布和不断地维护。

在软件测试的过程中会产生很多东西，比如测试的相关文档和测试各阶段的工作成果，包括测试计划文档、测试用例、还有自动化测试执行脚本和测试缺陷数据等。为了以后可能的查阅和修改，我们应该将这些工作成果和文档保存起来。

2. 测试配置管理的过程管理

了解了软件测试过程中配置管理的目标和阶段，接下来就应该进行软件测试配置过程管理了。软件测试配置管理过程包括如下内容。

1）建立配置变更控制委员会

配置变更控制委员会（CCB）应该要做到对项目的每个方面都有所了解，并且 CCB 这个团体不应该由选举产生，它的人员构成包括主席和顾问，在软件研发中每一个项目组都必须建立 CCB 作为变更权威。

2）SCM 库的建立和使用

我们要求在每一个项目过程中都要维护一个软件配置管理库。企业在项目中通过使用配置管理工具，简称（VSS），在配置管理服务器上建立和使用软件配置管理库。这些操作有助于在技术和管理这两个方面对所有的配置项进行控制，并且对它们的发布和有效性也能起到控制作用。同时还有很重要的一点就是我们应该对 SCM 库进行备份。这样做的目的是为了在产生意外或者风险时，能够作为保存灾难恢复备份的副本。

3）配置状态报告

配置状态报告是软件测试配置管理过程中的一项重要的活动，在软件测试配置管理过程中，配置人员要管理和控制所有提交的产品，然后在有产品提交或者变更为完成时，配置人员要经过相应的质量检查，这就是配置人员应该进行的工作。而在这之后，配置人员不但要将批准通过的配置项放入基线库中，并且还要记录配置项及其状态，编写配置状态说明和报告。通过配置人员的这些工作来确保所有应该了解情况的组或者个人能够及时地知道相关的信息。

4）评审、审计和发布过程

为了保持 SCM 库中内容的完整性和质量，我们应该采取适当的质量保证活动来应对 SCM 库中各项的变化。以此来确保在基线发布之前能够执行审计活动，该活动包括这几

点：基线审计、基线发布和产品构造。

软件测试过程中的配置管理就是由这些构成的。该过程不但提供给了我们良好的理论知识和清晰的认知，还让我们清楚地了解到软件测试过程中应该进行的工作有哪些。要想研发出好的软件需要进行好的软件测试。而要想进行好的软件测试，就需要我们掌握软件测试过程中的配置管理，并且了解该怎么样去运用它。只有对其有了深入的了解之后我们才能更好地进行软件测试工作，运用科学而且标准的测试配置管理知识为软件质量提供保障。

3. 测试配置管理的主要参与人员及其分工

了解了配置管理的目标和阶段以及如何进行软件测试配置过程管理，仅仅这些是不够的。有了这些知识，我们还不能够对软件测试进行完整的配置管理，不能仅凭借这些来有效地保障我们的软件质量。在这些基础之上，还需要对软件测试配置管理中的角色进行分配和分工。唯有如此才能确保在软件的开发和维护中，配置管理活动得到贯彻执行。因此在制定测试配置管理计划和开展测试配置管理之前，我们首先要确定配置管理活动的相关人员以及他们的职责和权限，下面我们来详细地了解配置管理过程中主要的参与人员和他们分职责分工。

1）项目经理（Project Manager，PM）

项目经理作为整个软件的开发以及整个软件的维护活动的负责人，那么他的职责是什么呢？他的主要职责包括采纳软件测试配置控制委员会的建议，对配置管理的各项活动进行批准，并且在批准之后还要控制它们的进程。项目经理的具体工作如下：首先制定项目的组织结构以及配置管理策略；然后要批准和发布配置管理计划；接着要对项目起始基线和软件开发工作里程碑进行制定；最后要接受并审阅配置控制委员会的报告。

2）配置控制委员会（Configuration Control Board，CCB）

该委员会的职责是对配置管理的各项具体活动进行指导和控制，并且为项目经理的决策提供建议。该委员会的具体工作职责如下：首先是要批准软件基线的建立以及配置项的标识；然后是制定访问控制策略；接着是建立、更改基线的设置和审核变更申请；最后是根据配置管理员的报告决定相应的对策。

3）配置管理员（Configuration Management Officer，CMO）

根据制定的配置管理计划执行各项管理任务这就是配置管理员的职责，配置管理员要定期向 CCB 提交报告，同时还要列席 CCB 的例会。他的具体工作职责如下：第一，对软件配置管理工具进行日常管理与维护；第二，提交配置管理计划；第三，各配置项的管理与维护；第四，执行版本控制和变更控制方案；第五，完成配置审计并提交报告；第六，对开发人员进行相关的培训；第七，对开发过程中存在的问题加以识别并制定解决方案。

4）开发人员（Developer，Dev）

开发人员的职责为在了解了项目组织确定的配置管理计划和相关规定之后，按照配置管理工具的使用模型来完成开发任务。

只有在清晰地了解了软件测试配置管理的概念、构成、原理和配置管理的人员及其分工之后，企业才能去灵活地应用它，在企业的软件测试过程中去严格地执行它。相信一个企业只要做好这一步，就一定能够做好软件测试工作，从而保证软件的质量，满足用户的需求。

7.5.3 测试配置管理的应用

下面以一个实际项目中的配置管理的例子来介绍项目中的配置管理应用。我们用来示例的项目是电信的一个项目，项目人员为16人，项目周期为一年，前期主要为开发、测试工作，后期主要是由维护人员进行系统维护和调整。在整个项目正式启动之前，配置管理工作就可以开始了。首先，应该评估团队当前的配置管理现状，清楚了解测试团队当前配置管理的现状是计划配置管理实施的基础，评估团队当前的配置管理现状有两种方法，一种是自己进行，另外一种是引入外部专业咨询人员来完成评估活动。有了评估的结果之后才能进行改进，因此，这项工作一定要做好。然后，定义实施的范围，在经过了评估之后，会找出很多的改进点，对于这些改进点，必须要花费大的精力来思考解决。还需要指定一个专门的人员就是配置管理员，在一个建立了配置管理平台的团队中他负责掌控整个团队的工作流程和成果，要负责维护和管理配置管理系统。有一个合格的配置管理员能够为整个团队的进度带来极其良好的影响。而在配置管理工作开始后的第一步就是制定一份配置管理计划。一般而言，需要在配置管理计划中明确的内容包括：

（1）配置管理软硬件资源；

（2）配置库结构；

（3）人员、角色以及配置管理规范；

（4）基线计划；

（5）配置库备份计划；

（6）执行配置审计。

下面我们来围绕其中一些内容进行详细描述。

1）配置管理环境

配置管理环境包括软硬件环境。具体的资源需求应该根据项目实际情况来确定，一般需要考虑的包括：网络环境、配置管理服务器的处理能力、空间需求、配置管理软件的选择等。配置管理环境的确定需要综合考虑各个方面的因素，包括采用的工具、人员对配置管理工具的熟悉程度等，同时，配置管理软件和测试工具的集成程度也是一个必须考虑的因素，根据经验，选择一个和测试环境集成紧密的配置管理工具，至少可以减少20%花费在Check In/Check Out和配置管理人员保持配置库完整上的工作量。

2）配置管理工具的选择

从测试人员具有的配置管理工具使用经验和配置管理工具使用的难易度方面来说，VSS是最好的选择，在现有的基础上只需要对测试人员进行简单培训；考虑到和测试工具的集成，VSS也是一个不错的选择。不过本项目还要求对远程接入方式的支持，以及对Solaris平台的支持，VSS肯定是不能满足要求的（VSS通过VPN方式应该是可以实现对远程访问的支持，但VSS的完全共享方式实在是不敢在Internet上使用）。除VSS外，可以选择的配置管理工具还有CCC Harvest、ClearCase、CVS等，但Harvest和ClearCase使用起来比较复杂，需要一个专门的配置库管理员负责技术支持，还需要对测试人员进行较多的培训。

3）配置库维护和备份计划

配置库的维护和备份需要专职的配置库管理员来负责。在整个项目中我们采用的配置

库维护策略是根据 Microsoft 的 Best Practice 白皮书建议，包括以下要点：

(1) 保持配置数据库的大小不超过 5G；Microsoft 建议，配置库的大小在 3G～5G 比较合适，太大的数据库会极大影响 VSS 的效率。

(2) 每周进行 VSS 数据库的分析(Analysis)，发现问题及时修正；VSS 提供了 Analysis 和 Fix 工具，由于不合理的 Delete 等操作，VSS 数据库有可能会出现一些 Interrupt Data 之类的问题，通过定期的每周的分析工作，可以极大减少数据库出现问题的风险。

(3) 每日进行配置库的增量备份，每周进行数据库的完全备份；VSS 库的备份可以通过 VSS 自己的 Archive 功能或者是操作系统的 Backup 程序来进行。VSS 的 Archive 功能对 VSS 中的文件数据进行压缩并保留 VSS 的所有状态，但只能对 VSS 库进行完全备份，不能实现增量备份功能。Windows 2000 Server 提供的 Backup 实用程序可以对文件进行备份，由于 VSS 库就是以文件形式存在的，因此针对 VSS 的 data 目录进行备份也可以完全达到备份的目的，使用系统备份工具的好处是可以实现增量备份。比如我们在实际中使用系统的备份工具，每周五生成的完全备份采用刻录光盘的方式保存，每天的增量备份数据存放在文件服务器上进行备份。

7.5.4　软件测试的测试版本控制

通过上面我们对软件测试的配置管理有了详细的了解，此外我们还需要了解配置管理中的另一方面，那就是软件测试过程中的版本控制，这同样也是软件测试过程中不可缺少的一部分。很多人不了解软件测试过程中的版本控制，甚至认为软件测试不需要版本控制。这种想法是错误的，在软件测试中也同样需要版本控制，这是软件测试中不能缺少的一部分。如果测试组长或者测试人员不对软件测试进行版本控制，那么带来的危害也是显而易见的。软件测试过程中如果缺乏版本控制，很难保证测试进度和测试的一致性。我们知道在进行测试工作时，很容易出现的是冗余这一问题，这样很容易导致本地版本和服务器版本的不一致，缺乏版本控制就会造成上面这些问题。由此可见，软件测试是离不开版本控制的。在测试过程中适合的版本控制可以有效地提高开发和测试效率，消除很多由于版本带来的问题，并且可以确保在软件开发和测试过程中，能够及时并且正确地更新不同的人员所涉及的同一文档。

7.5.5　测试版本控制的概念

软件测试的版本控制简单地说就是对测试有明确的标识、说明，并且测试版本的交付是在项目管理人员的控制之下。用来识别所用版本的状态就是对测试版本的标识，对不同的版本进行编号，软件质量稳定度趋势的反映也可以由测试的版本控制体现。版本控制是软件测试的一门十分实用的实践性技术，将各次的测试行为以文件的形式进行记录，并且对每次的测试行为进行编号，标识公布过的每一个测试版本，以此来进行测试排序，比如将最初的版本标识为 1，经过测试后，之后的版本依次标识为 2、3、4 等。

1. 测试版本控制的作用

测试版本控制有两个方面的作用：一方面是标记历史上产生的每个版本的版本号和测

试状态，另一个方面是保证测试人员得到的测试版本是最新的版本。所谓版本控制其实就是跟踪标记测试过程中的软件版本，以方便对比的一个过程，通过版本控制来表明各个版本之间的关系和不同的软件开发测试阶段，从而方便测试工作的进行。版本控制是测试人员不可缺少的一种技术。有了版本控制，测试人员的软件测试工作可以更加高效并且针对性地进行。

2. 如何做好测试版本控制

测试版本控制其实是在软件测试中为了便于追溯和跟踪问题而出现的，对测试版本的控制实际上就是对软件测试过程中的各种测试行为的管理和控制。软件测试的版本控制，主要是指对于测试对象的版本控制，也就是指测试小组在软件测试过程中对开发部提交给测试部门的产品进行版本控制。如果开发小组不能够规范管理软件版本，那么这时候测试小组对于产品进行版本控制就将显得十分重要，这时软件测试小组要保证测试对象的可控性被限制在我们可以控制的范围之内。对此，我们建议开发小组和测试小组要做到如下要求：两个小组不但分工明确，还要协商出一个明确的约定，指定专门的测试版本负责人来专门负责版本控制这一块，让这个负责人去制定版本控制的提交原则，在软件研发过程中对提交的情况要进行详细的记录，通过这些措施，基本上能避免因为版本失控可能造成的测试失误或者测试无效。

举一个软件测试项目中版本控制的例子，一个公司的员工负责了一个软件测试项目，在项目开展初期，该项目的测试工作进行得还算顺利。但是在测试后期工作即将结束时，却出现了问题。原因在于在这个软件测试项目中，他们将测试过程中发现的 Bug 提交给开发人员，开发人员再对测试人员提交的 Bug 进行修改，但直接将修改后的代码放入了当前的软件版本之中，问题正是出现在这个阶段。那么为什么问题会出现在这个阶段呢？其实是因为对于修改过的代码，我们不能够保证它们一定是正确的，很可能在开发人员修改过之后，仍然是错误的，或者在修改过之后仍然会给软件带来其他问题，这种情况下就会给软件测试人员的测试工作带来新的麻烦。这样就会造成一个很严重的后果，那就是测试人员对开发人员提交的新代码会很紧张，不能彻底放心，测试人员对新提交的新修正的代码还要再进行验证，进行排错，来确保不会因此而带来新的隐患、新的漏洞。在这个时候我们就应该思考怎样去解决这个问题了，这时我们就可以尝试一下软件测试的版本控制。首先测试人员要测试开发人员提交的代码，将测试过程中查找到的 Bug 进行提交。而当测试人员提交的 Bug 到了开发人员手中之后，开发人员要针对这些 Bug 进行修复工作，并且将修改后的代码放入程序中，作为新的软件版本。但是绝对不能将它再放回到现在正在进行的测试版本中。而测试人员在完成这一轮的测试工作后，再对新的版本也就是对经过开发人员修改过的下一个版本展开新一轮的测试。这就是软件测试过程中的版本控制。

3. 缺乏测试版本控制的危害

软件测试缺乏测试版本控制会带来很多危害：随着计算机软件技术的日趋成熟和日益复杂。现代的软件产品规模越来越大，结构越来越复杂。单纯的手工测试不再能够满足软件工程中的测试需要。一两个人完成一个项目的测试工作的时代不复存在，一个大型项目的测试工作，现在都是由很多测试人员参与并且有不同的分工，以团队合作流水线式的方式

来开展工作的，他们在测试过程中是协同完成测试工作的。在进行测试的时候，同一个板块的测试会由很多人共同负责，他们将会分配到同样的任务。在这么多人完成同一板块的过程中，企业怎样去保证每个人的测试工作对软件产生的影响能够综合到一起产生好的作用，而不是因为人员的差异而对测试版本造成不一致问题呢？运用软件测试中的版本控制就成为此时有效的解决途径，有效的版本控制能够很好地解决这些问题，并对软件的开发进行产生好的积极的影响。但是如果版本控制不当则会造成很多让人棘手的问题。

1）缺乏版本控制，将难以保证测试进度

大多数的测试人员都希望他们进行的测试工作是完美的，经过他们测试后的软件更是完美无缺的。这种想法是好的，但是一个软件在它的整个生命过程中是不可能完美到没有一点错误存在的，我们只能尽所能去不断地完善它。这时如果能够提供有效的版本控制就会极大地提高软件测试的工作效率。对测试进行版本控制时，我们起码要做到能够掌握软件过程中的每个版本。而在每个版本中，我们能够找到哪些功能不过关，哪些功能没问题。对于新的测试版本不管在测试工作中的哪个时间段，都应该有一个可以用来比较的对象，并且能够与之前的版本进行对比。

2）缺乏版本控制，难以保证测试的一致性

软件测试工作十分复杂并且有很多人员参与其中，因此必然要对测试人员进行分工，不同的测试人员又要负责不同的测试模块。但是因为软件有其整体性，所以测试人员又要互相协作，那么在测试过程中则必然会产生交叉。在整个测试中为了保证测试过程中的一致性我们必须要找到一个平衡点。而大量的实践证明如果进行有效的版本控制，将有效保证测试过程中的一致性，大大降低因为缺乏版本控制或者流程管理可能带来的诸多问题。

3）测试版本冗余，易出现误用风险

因为有众多测试员参与到软件测试过程中，而进行测试工作时每个人都必须使用一台计算机，那么在每个人的计算机上都要复制一个待测软件。随着测试工作的进行，在测试过程中会不断产生新的软件版本，为了测试需要，每个人的计算机上都要不断更新软件版本，那么每个人的计算机上必然会保存不同时期的软件版本。而这些不同的测试版本随着时间的推移很容易混杂在一起，造成测试人员无法分清每个版本之间的差异，甚至分不清对于当前版本应该做什么事情，从而给测试工作带来极大的困扰，出现版本的冗余。这时，如果缺乏有效的测试版本控制就会增大测试风险，给测试工作带来麻烦。

4）容易导致本地版本和服务器版本不一致

因为测试版本的众多和混乱，测试小组在测试工作上不但要花费更多的时间和精力，还可能造成重复测试和不必要测试，而且当测试版本不及时更新时，会造成测试版本和现行版本的不一致。因而加强测试过程中的版本控制是一项很重要的工作。

5）缺乏测试文档可追溯性

版本控制在提供可追溯性的文件的同时，还能够为各种测试版本提供文档管理支持，使我们能够很方便地随时查阅在软件测试过程中生成的各种文档。

4．测试版本控制方法及工具解析

1）测试版本控制的方法

有效的版本控制能够极大地方便软件测试工作，提高测试工作的效率，那么我们如何才

能成功进行版本控制呢？为此我们应该制定一套标准，制定相应的版本控制方法来规范化软件测试的版本控制。方法如下：

① 在软件测试过程中制定规范的版本控制管理制度，明确整个测试中的测试需求，选择合适的版本控制切入点，把版本控制和测试里程碑结合到一起来实现阶段性成果，从而避免测试规划过程混乱的风险。

② 通过制定合理的版次规划和监控机制来进行版本控制，为了有效地管理测试项目所需的版本次数，应该对测试工作量进行合理评估，以此来做出合理的版次规划。

③ 不能忽略版本控制管理员在版本控制中的重要性，版本控制管理员在测试版本控制中的重要性是不可估量的，离开了版本控制管理员和缺乏版本控制的情况是等效的。

④ 要做好版本控制的文档管理，对相关文档进行严格规范的管理，将测试过程中版本控制产生的相关文档记录、标识是很重要的，有了这些能够很方便地跟踪和监控测试版本的执行。

⑤ 选择合理的应用版本控制的软件工具，能够极大提高测试工作的效率，大大提高测试活动的优质性。

2）测试版本控制工具解析

软件测试中版本控制的重要性已无须再多言，为了有效地进行版本控制，选择一款好的测试工具这时就显得尤为重要。下面简单介绍一款版本控制工具 ClearCase，同时它也是一种配置管理的工具。它是由 Rational 公司开发的一款配置管理工具。ClearCase 的四种功能如下：

① 控制任何文件的版本（Version Control），它能够维护和控制软件版本，有效管理版本内容。

② 在版本树中组织元件发展的过程（Workspace Management）。针对目录结构它可以定制一个版本树的结构，并且其中包含多层分支和子分支。

③ 对目录和子目录进行版本控制（Build Management），比如，在其中建立一个新的文件夹，或者对文件名进行修改，又或者新建子目录或者在不同的目录间移动文件等。

④ 明确项目设计的流程（Process Control）。比如，可以通过将不同的权限授权给全体人员来阻止某些修改的发生，立刻通知团队成员任何时刻某一事件的发生，对开发的进程建立一个永久记录并不断维护它。

5. 测试版本控制应用

下面以一个实际项目中的例子来介绍项目中的版本控制。在这个项目中，我们已经分好了测试小组及组内成员的分工，启动软件测试的条件已经具备，在开发人员发布了测试版本后，有相应的文档支持比如自测报告、软件版本说明等。然后启动测试工作，其中，测试小组对项目进行版本控制分以下几步：第一，制定规范的版本控制管理制度，测试小组要了解和明确整个测试的需求，选择合适的版本控制切入点，对于测试中产生的测试版本要进行严格控制，还要规范化控制测试过程中产生的不同时期的测试版本，通过把版本控制与测试里程碑结合来实现阶段性成果，以规避测试过程混乱的风险。第二，应该制定合理版次规划和监控机制，对测试项目所需的版次数量进行有效管理，并且对版次做出合理的规划。在整个测试项目的关键位置要设立检查点，要能根据版次规划随时监控版本更新，及时发现问题，

并对出现的异常现象做出快速反应，这样才能使得测试过程更加清晰和更有计划性。第三，测试小组要指定版本控制管理员。测试版本控制作为一个贯穿整个测试周期的一项活动，测试版本控制会涉及到很多的人员角色，其中最为主要的人力资源就是测试版本控制管理员。在一个建立了良好版本控制机制的测试团队中测试版本控制管理员是十分重要的。他负责掌控整个测试过程的测试版本，要负责记录和监控测试中不同测试阶段产生的测试版本。他的地位是不可动摇的。有一个合格的测试版本控制管理员能够为版本控制工作带来极其良好的影响。最后是选择一款合适的版本控制工具，在进行版本控制的过程中测试小组选择一款合适的测试版本控制软件是不可缺少的。常见的版本控制工具有 CVS、SVN、ClearCase，其中 CVS 是一款开放源代码软件，其功能强大，跨平台、支持并发版本控制而且免费，所以它在中小型软件企业中得到广泛使用。但是它最大的遗憾就是缺少相应的技术支持，许多问题的解决需要自己寻找资料，甚至是研究源代码。SVN 则是针对 CVS 的缺点进行改进产生的版本控制工具，相比 CVS 而言，SVN 更为简单易用，且 SVN 有其特定平台的客户端工具。如 TortoiseSVN，是为 Windows 外壳程序集成到 Windows 资源管理器和文件管理系统的 SVN 客户端，使用相当方便。最后一种版本控制工具 ClearCase 是 Rational 公司一款重量级的软件配置管理工具。与 CVS 和 SVN 不同的是，ClearCase 涵盖的范围包括版本控制、建立管理、工作空间管理和过程控制。ClearCase 贯穿于整个软件生命周期，支持现有的绝大多数操作系统，但它的安装、配置、使用相对较复杂，需要进行团队培训。选择一款合适的版本控制工具不但能够提高测试工作效率，而且也会大大提高测试活动的优质性。

7.6　软件测试风险管理

7.6.1　风险管理

项目的未来充满风险。风险是一种不确定的事件或条件，一旦发生，会对至少一个项目目标造成影响，如范围、进度、成本和质量。风险可能有一种或多种起因，一旦发生可能有一项或多项影响。风险的起因包括可能引起消极或积极结果的需求、假设条件、制约因素或某种状况。

项目风险管理包括风险管理规划、风险识别、风险分析、风险应对规划和风险监控等各个过程。项目风险管理的目标在于提高项目积极事件的概率和影响，而降低项目消极事件的概率和影响。对于已识别出的风险，需要分析其发生概率和影响程度，并进行优先级排序，优先处理高概率和高影响风险。

7.6.2　风险识别

风险识别旨在系统化地识别已知的和可预测的风险，只有识别才能提前采取措施，尽可能避免这些风险的发生，最重要的是量化不确定性的程度和每个风险可能造成损失的程度。

风险可分为以下几种类型：

(1) 需求风险，如需求变更频繁、缺少有效的需求变更管理；

(2) 计划风险,如实际规模比估算规模大很多、项目交付时间提前但没有调整项目计划;

(3) 人员风险,如项目新员工较多、骨干员工不稳定;

(4) 环境风险,如设备未及时到位、新开发工具学习时间较长、环境未及时到位;

(5) 产品风险,如新产品、新技术、基础版本质量不高;

(6) 客户风险,如客户问题确认时间过长、客户不能保证投入需求评审;

(7) 组织和管理风险,如低效的项目团队结构降低生产率、缺乏必要的规范,导致工作失误与重复工作;

(8) 过程风险,如前期质量保证活动执行不到位,导致后期的返工工作量过大、需求方案确认时间过长。

风险识别主要有以下方法:

(1) 头脑风暴:组织测试组成员识别可能出现的风险;

(2) 访谈:找内部或外部资深专家访谈;

(3) 风险检查列表:对照表的每一项进行判断,逐个检查风险。

7.6.3 风险评估

风险评估是对已识别风险的影响和可能性大小的分析过程。从经验来看,许多最终导致项目失败、延期、客户投诉的风险,都是从不起眼的小风险开始,由于这些小风险长时间得不到重视和解决,最终严重影响到项目交付。

风险评估的主要任务包括:

(1) 评估对象面临的各种风险;

(2) 评估风险概率和可能带来的负面影响;

(3) 确定组织承受风险的能力;

(4) 确定风险消减和控制的优先等级;

(5) 推荐风险消减对策。

风险评估要重点关注以下几个方面:

(1) 风险的性质,即风险发生时可能产生的问题;

(2) 风险的范围,风险的严重性及其总的分布;

(3) 风险的时间,何时能感受到风险及风险维持多长时间。

7.6.4 风险应对

风险应对是对项目管理者管理水平的最好检验,从风险预防、识别、评估到应对措施及结果,能检验出管理者的综合水平。在项目过程中,风险应对不是简单地消除风险。

风险应对的主要方法如下:

(1) 规避风险,主动采取措施避免风险,消灭风险;

(2) 接受风险,不采取任何行动,将风险保持在现有水平;

(3) 降低风险,采取相应措施将风险降低到可接受的水平;

(4) 风险转移,付出一定的代价,把某风险的部分或全部消极影响连同应对责任转移给第三方,达到对冲项目风险的目的。

风险识别及监控主要形式有：

(1) 使用风险管理表单跟踪每一个风险，定期核对各风险发生的紧急程度；

(2) 通过晨会、日报、周报、周例会等形式从团队内部出发识别出新的风险，反馈风险处理情况；通过与客户沟通、上级汇报等形式，以团队外部评价收集风险信息。

7.7　软件测试的成本管理

项目的各种管理从时间上看都有一个开始阶段、中间阶段和结束阶段。项目的生命周期都会经历初始阶段、计划阶段、执行阶段、控制阶段和结束阶段。其中计划、执行和控制是一个循环反复的过程，制订计划，按照计划执行，执行中进行控制，不行则返回修改计划，修改后继续执行，直到成功完成该项目。项目的核心过程有范围管理、时间管理和成本管理。下面就成本管理展开讨论有关知识、经验。

成本管理是为了保证项目能够在核定的预算下完成。成本管理包括资源计划、成本估计、成本预算核定和成本控制。成本管理的每一部分都有输入、工具技术和输出。资源计划是根据 WBS、历史信息、范围陈述、资源池描述、组织方针和活动持续期预计，利用专家判断、选择性鉴定和项目管理软件，得到资源需求文档。成本估计是根据 WBS、资源需求说明、资源费用、活动持续期估计、估计发布和历史信息及账目表、风险，利用相似估计、参变模型、自底向上估计、计算机化工具和其他成本估计方法，得出成本估计、支持细节和成本管理计划。成本预算核定是根据成本估计、WBS、项目进度和风险管理计划，利用成本预算工具和技术，得到项目成本基线。(成本基线是基于有限时间的预算，常用来测量监视项目成本性能。)成本控制是根据成本基线、性能报告、需求变化和风险管理计划，采用成本变化管理系统、性能测量、增值管理、附加计划和计算机化工具，得到修正的成本估计、预算变动、纠正活动和完成估计(EAC)。

每个项目都可根据一定的原则分为一系列活动，每一活动还可以分为一系列的子活动，一级级地划分直到不能或不需要划分为止。如某种材料，某种设备，某一活动单元等。然后估算每个 WBS 要素的费用。采用这一方法的前提是：

(1) 对项目的需求有一完整的限定；

(2) 制定完成任务所必需的逻辑步骤；

(3) 编制 WBS 表。

项目需求的完整限定应包括工作报告书、规格书以及总进度表。工作报告书是指实施项目所需的各项工作的叙述性说明，它应确认必须达到的目标。如果有资金等限制，该信息也应包括在内。规格书是对工时、设备以及材料标价的根据。它应该能使项目人员和用户了解工时、设备以及材料估价的依据。总进度表应明确项目实施的主要阶段和分界点，其中应包括长期定货、原型试验、设计评审会议以及其他任何关键的决策点。如果可能，用来指导成本估算的总进度表应含有项目开始和结束的日历时间。

一旦项目需求被勾划出来，就应制定完成任务所必需的逻辑步骤。在现代大型复杂项目中，通常是用箭头图来表明项目任务的逻辑程序，并以此作为下一步绘制 CPM 或 PERT 图以及 WBS 表的根据。

编制 WBS 表的最简单方法是依据箭头图。把箭头图上的每一项活动当作一项工作任

务，在此基础上再描绘分工作任务。

进度表和 WBS 表完成之后，就可以进行成本估算了。成本估算的结果最后应以下述的报告形式表述出来。

(1) 对每个 WBS 要素的详细费用估算。还应有一个各项分工作、分任务的费用汇总表，以及项目和整个计划的累积报表。

(2) 每个部门的计划工时曲线。如果部门工时曲线含有“峰”和“谷”，应考虑对进度表作若干改变，以得到工时的均衡性。

(3) 逐月的工时费用总结。以便项目费用必须削减时，项目负责人能够利用此表和工时曲线作权衡性研究。

(4) 逐年费用分配表。此表以 WBS 要素来划分，表明每年(或每季度)所需费用。此表实质上是每项活动的项目现金流量的总结。

(5) 原料及支出预测，它表明供货商的供货时间、支付方式、承担义务以及支付原料的现金流量等。

划分项目的 WBS 结构有许多方法，如按照专业划分、按照子系统、子工程划分、按照项目不同的阶段划分等，以上每一种方法度有其优缺点。一般情况下，确定项目的 WBS 结构需要组合以上几种方法进行，在 WBS 的不同层次使用不同的方法。良好的项目管理必须具备以下因素：对项目的认知、为项目提供良好的协同环境和有效的控制。这几个因素环环相扣，前者是后者的必要条件。一个良好的 WBS 结构在项目管理中所起的作用也可以这三个层次来理解。

首先是按照专业划分项目，应当说这是一种最自然的划分方法，优点是容易让人接受，缺点是不易协调。比如，在进行地铁建设时，假定在 WBS 的顶层按照专业将建设分为土建和安装，并按照这种划分确定一个土建分项目经理和一个安装分项目经理。按照这种划分方法在画项目的网络图时就会出现一系列的土建作业和一系列的安装作业。因为某一个车站既包括土建工程又包括安装工程，这样在两组作业组之间就会出现非常复杂的关系，分项目经理之间也很难协调工作。按照系统划分方法容易界定项目范围，但有时候显得不那么直观。系统是人们在长期实践中确定的一种分类方法，其特点是系统与系统之间的联系往往是比较简单的，这种联系通常被称为系统界面或接口。正由于系统之间的界面比较清楚，所以按照系统对项目进行划分更容易界定子项目或子工程的范围，在项目实施过程中更容易控制结果。按照项目的不同阶段划分 WBS 结构有利于项目管理者控制中间结果。对那些不确定性比较大的项目来说，项目最后的结果往往是未知的，控制项目的唯一方法就是控制中间结果的进度和质量，当然阶段的划分应该是可测量的。按照阶段划分项目有助于管理者在不同阶段控制中间成果同时不至于使项目管理者陷入到项目细节中。不同的项目，其范围、性质可能都不一样，项目管理的目标和重点不尽相同，项目的 WBS 结构也并不一样。但无论对何种项目进行 WBS 划分，都必须兼顾 WBS 的三种不同层次的作用。划分项目的 WBS 结构还必须遵循一定的方法论。具体说来，划分项目的 WBS 结构必须遵循以下步骤：

(1) 确定项目特性并确定 WBS 层次，比如项目的不确定性有多大？项目的规模又是多大？

(2) 确定项目管理的重点，为项目管理目标划分优先级别，比如，项目质量是放在第一

位的，还是项目进度居于首位？

（3）针对项目管理目标的优先级别确定每级 WBS 划分方法。

（4）确定 WBS 结构。

下面列举一些针对软件企业的有效成本控制的经验。

软件企业必须加大宣传力度，树立全员经济意识。首先要统一思想认识，从项目管理人员到普通员工要进行经济教育，灌输经济意识，把一切为了效益的意识深深地刻在每个职工的脑海里，诸如“节约光荣，浪费可耻”等，使每一位职工都能把工程成本管理工作放在主要位置。其次是组织培训，提高专业人员的素质，这是实现成本目标的保证。

建立一套行之有效的制度并非易事，每一个工程项目都有其自身的特点，要根据工程项目本身的特点，制定有针对性的项目成本管理办法，如项目质量成本管理办法、工期成本管理办法、项目招投标管理办法、合同评审管理办法、材料使用控制办法等管理办法。这些管理办法应是责任到人、切实可行的具有较强操作性的办法，使项目的成本控制有法可依、有章可循、有据可查。项目成本实施的主体是项目组人员，项目经理是项目成本管理的领导，这样形成了一个以项目经理为核心的成本管理体系。对成本管理体系中的每个部门、每个人的工作职责和范围要进行明确的界定，遵循民主集中制、标准化、规范化的原则进行建立职责权利相结合的成本管理模式和体制；对于每个项目，都要有成本控制的目标——项目预算，都要严格按照规范做 WBS(工作任务分解)，在落实任务的同时，也要落实完成任务需要的成本预算，并且逐级负责，层层落实，使项目成本管理工作做到责权利无空白，无重叠，事事有人管，责任有人担，一切有章可循，有据可查，杜绝了推诿扯皮，使项目的成本管理工作形成一个完整的成本管理体系，同时用一定物质奖励去刺激，使每个人的工作、成本和项目的效益挂钩，彻底打破过去那种干好干坏一个样，干多干少一个样的格局，调动职工的积极性和主动性，使大家共同为项目的成本管理献计献策。

7.8　本章小结

测试项目管理就是以测试项目为管理对象，通过一个临时性的专门的测试组织，运用专门的软件测试知识、技能、工具和方法，对测试项目进行计划、组织、执行和控制，并在时间成本、软件测试质量等方面进行分析和管理活动。本章从软件测试项目管理概述、软件测试文档、软件测试的组织与人员管理、软件测试过程管理、软件测试的配置管理、软件测试风险管理、软件测试的成本管理等几个专题讨论了对测试项目的全过程进行管理。

习题 7

1. 测试项目管理的基本特征是什么？
2. 主要软件测试文档有哪些，它们分别都有哪些作用？
3. 软件测试中如何来进行变更控制？
4. 简述测试配置管理的方法和内容。
5. 软件测试如何来进行风险管理？

第8章 面向对象软件测试

俗话说，思维方式决定解决问题的方式。在传统软件开发中采用面向过程、面向功能的方法，将程序系统模块化，并在此基础上再分成若干个单元，这些单元可以通过一系列程序得到实现，由此产生了相应的单元测试、集成测试等方法。面向对象程序的结构不是传统的功能模块结构，它将开发过程分为面向对象分析（OOA）、面向对象设计（OOD）和面向对象编程（OOP）三个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。针对面向对象软件的开发特点，测试方法和技术也必然要做相应的改变，因而形成了面向对象的测试模型、测试的层次与数据流，以及面向对象的单元和集成测试方法等，这些都是本章要介绍的内容。

8.1 面向对象软件的特点及其对测试的影响

我们生活在一个对象的世界里，每个对象有一定的属性，把属性相同的对象进行归纳就形成类。例如，家具就可以看做类，其主要的属性有价格、尺寸、重量、位置和颜色等。无论我们谈论桌子、椅子，还是沙发、衣橱，这些属性总是可用的，因为它们都是家具，继承了为家具类定义的所有属性。实际上，计算机软件所创建的面向对象思想同样来源于生活。

面向对象技术是一种全新的软件开发技术，逐渐代替曾被广泛使用的面向过程开发方法。面向对象技术产生更好的系统结构，更规范的编程风格，极大地优化了数据使用的安全性，提高了程序代码的重用。

面向对象程序设计的核心是对象。在面向对象程序设计中，对象是实现世界中各种实体的抽象表示，它是数据和代码的组合，有自己的状态和行为。具体来说，对象的状态用数据来表示，称为对象的属性，而对象的行为用代码来实现，称为对象的方法，不同的对象会有不同的属性和方法。

类是具有相同数据类型和相同操作的一组对象的集合，它是对具有相同属性和行为的一组相似对象的抽象。

类描述了属于该类型的所有对象的特征和行为信息，是生成对象的蓝图和模板。类通过设定该类中每个对象都具有的属性和方法来提供对象的定义，也就是说有关对象的属性、方法和事件是在定义类时指定。每一个属于某个类的特定对象称为该类的一个实例。创建了一个类后，可以创建所需的任何数量的对象。

面向对象可以定义为：面向对象（object oriented）＝对象＋分类＋继承＋通信。

面向对象技术促成程序构件的复用，而复用促成更快的软件开发和高质量的程序。面向对象软件易于维护，因为它的结构是内紧外松，这样当进行修改时，影响面小。此外，面向对象系统易于进行适应性修改及伸缩。归纳起来其优点有如下几个：

（1）可重用性

从一开始对象的产生就是为了重复利用，完成的对象将在今后的程序开发中被部分或全部地重复利用。

（2）可靠性

由于面向对象的应用程序包含了通过测试的标准部分，因此更加可靠。由于大量代码来源于成熟可靠的类库，因而新开发程序的新增代码明显减少，这是程序可靠性提高的一个重要原因。

（3）连续性

具有面向对象特点的 C++与 C 语言有很大的兼容性，C 程序员可以比较容易地过渡到 C++语言开发工作。

面向对象程序与传统面向过程程序的一个主要区别在于：面向过程的程序鼓励过程的自治，但不鼓励过程间交互；面向对象的程序则不鼓励过程的自治，并且将过程（即方法）封装在类中，而类的对象的执行则主要体现在这些过程的交互上。

与传统的程序相比较，面向对象程序主要特性有：封装、抽象、继承、多态。

1. 对象的封装和抽象

为有效使用面向对象的程序方法，首先需要解决程序的结构设计问题。在程序设计过程中最重要的是抽象，也就是说，从现实世界中抽象出合理的对象结构。在面向对象思想中.抽象决定了对象的属性、内部结构以及处理对象的外部接口。

从程序语言角度来看，在一个对象中代码和（或）数据可以是这个对象私有的，不能被对象外的部分直接访问。因而对象提供了一种高级保护以防止程序被无关部分错误修改或错误地使用了对象的私有部分。当从对象外部试图直接对受保护的内部数据进行修改时，将被程序拒绝，只有通过对象所提供的对外服务函数才能够对其内部数据进行必要的加工，从而保证了数据加工的合法性。从这一意义上讲，把这种代码和数据的联系称为“封装”。换句话说，封装是将对象封闭保护起来，是将内部细节隐蔽起来的能力。

2. 继承性与多态性

面向对象设计方法的另一个重要贡献是关于继承性与多态性的处理。所谓继承是指从已经存在的对象出发建立一个新的对象类型，使它具有原对象的特点和功能。同时，新的对象类里又具有某种新特点和新功能。这样可以采用对象继承的方法建立一个有层次的对外部世界的描述。例如，我们可以想象有一组三维曲面分块的类族，从四条三维空间直线定义的简单曲面，到四条复杂三维空间曲线定义的曲面，呈现复杂的分层次的多态性，但它们都有共同的接口函数形式。

- 访问控制。对象必须能够对其内部的某些元素进行保护，使它们只能被内部使用，而不受外部干扰。反过来，对象又必须同其他外部元素进行联系，以便对对象进行操作。在 C++中，类分为私有的（private）、保护的（protected）和公有的（public）三

种访问机制。

- 继承性。可通过对已有对象进行增加或部分修改的方法来建立新的对象，对已有对象可以增加数据和过程，也可以对其中某些过程进行重新定义。最初的类被称为基类，从基类扩展出来的类称为派生类。从已有类派生出新类是为了获得更强的针对性。
- 多态性。正像生态系统一样，继承的出现使得类簇得以诞生。通常这些类簇中的类具有同名的成员函数，例如，OD 分布类簇，具有一个通用基类、两个派生类——增长系数 OD 分布类和重力模型 OD 分布类，这几个类都有同名的 Exec 成员函数。多态性意味着存在多种形式，能使人们在程序中激活任何属于 OD 分布类簇的 Exec 成员函数，甚至在编译可以不必具体知道对象是属于增长系数 OD 类还是重力模型 OD 分布类。

8.2 面向对象软件测试的不同层次及其特点

一般来说，面向对象软件的测试可分为三或四个层次。

这里主要取决于对单元的构成，若把单个操作和方法看作单元，则有四个层次。

(1) 方法测试：指对类中的各个方法进行单独的测试。

(2) 类测试：类测试的重点是类内方法间的交互和其对象的各个状态。

(3) 类簇测试：类簇也叫子系统，由若干个类所组成，类簇测试重点是测试一组协同操作类之间的相互作用。

(4) 系统测试：系统测试检验所有类和整个软件系统是否符合需求。

三个层次方式以类为单元，这样对标识测试用例非常有利，同时使得集成测试有更清晰的目标。如下所示：

(1) 面向对象单元测试是进行面向对象集成测试的基础。

(2) 面向对象集成测试主要对系统内部的相互服务进行测试。

(3) 面向对象系统测试是基于面向对象集成测试的最后阶段的测试。

下面以三个层次为例进行详细描述。

1. 面向对象的单元测试——类测试

面向对象软件的类测试相当于传统软件中的单元测试。类的测试用例可以先根据其中的方法设计，然后扩展到方法之间的调用关系。类测试一般也采用功能性测试方法和结构性 测试方法，传统的测试用例设计方法在面向对象单元测试中都可以使用，例如，等价类划分法、因果图法、边值分析法、逻辑覆盖法、路径分析法等。

功能性测试以类的规格说明为基础，主要检查类是否符合其规格说明的要求。功能性测试包括两个层次：类的规格说明和方法的规格说明。

结构性测试则是从程序出发，对类中方法进行测试，需要考虑其中的代码是否正确。测试分为两层：第一层考虑类中各独立方法的代码，即方法要做单独测试；第二层考虑方法之间的相互作用，即方法需要进行综合测试。

面向对象编程的特性使得对类中成员函数的测试又不完全等同于传统的函数或过程测试。尤其是继承特性和多态特性，使子类继承或重载的父类成员函数出现了传统测试中未

遇见的问题。这里要考虑如下两个问题。

(1) 继承的成员函数是否都不需要测试？

对父类中已经测试过的成员函数，以下两种情况需要在子类中重新测试：

- 继承的成员函数在子类中做了改动。
- 成员函数调用了改动过的成员函数的部分。

例：假设父类 Base 中有 InheritedK()和 RedefinedK()这两个成员函数，继承 Base 的子类 Derived 只对 Redefined()做了改动。那么，Derived::Redefined()就需要重新测试；对于 Derived::InheritedK()，若它包含了调用 Redefined()的语句(比如，x = x/Redefined())，就需要重新测试，否则就不需要。

(2) 对父类的测试是否能照搬到子类？

引用前面的假设，成员函数 Base::RedefinedK()和 Derived::Redefined()已经是不同的。那么，按理应该要对 Derived::Redefined()重新测试，需要分析和设计测试用例。但是由于面向对象的继承使得两个函数相似，故只需要在对 Base::RedefinedK()的测试要求和测试用例上添加对 Derived::RedefinedK()的新测试要求和增补相应的测试用例。

例如，Base ::Redefined()含有如下语句：

```
If (value < 0) message("less");
 else if (value == 0) message( "equal");
else message( "more");
```

在 Derived :: Redefined ()中重定义为：

```
If (value < 0 ) message("less");
else if (value == 0) message("It is equal");
else {
message("more");
if (value == 88) message("luck");
}
```

对 Derived::Redefined()的测试只需在原有对父类 Base 的测试上，作如下改动：将 value = =0 的测试结果期望改动；增加对 value ==88 的测试。

面向对象设计方法通常采用状态转移图建立对象的动态行为模型。状态转移图用于刻画 对象响应各种事件时状态发生转移的情况，图中节点表示对象的某个可能状态，节点之间的有向边通常用“事件/动作”标出。基于状态的测试是通过检查对象的状态在执行某个方法后是否会转移到预期状态的一种测试技术。使用该技术能够检验类中的方法是否能正确地交互。因为对象的状态是通过对象数据成员的值反映出来，所以检查对象的状态实际上就是跟踪监视对象数据成员的值的变化。如果某个方法执行后对象的状态未能按预期的方式改变，则说明该方法含有错误。基于状态测试的主要步骤如下：

(1) 依据设计文档，或者通过分析对象数据成员的取值情况空间，得到被测试类的状态转移图。

(2) 给被测试的类加入用于设置和检查对象状态的新方法，导出对象的逻辑状态。

(3) 对于状态转移图中每个状态，确定该状态是哪些方法的合法起始状态，即在该状态时对象允许执行哪些操作。在每个状态，从类中方法的调用关系图最下层开始，逐一测试类

中的方法；测试每个方法时，根据对象当前状态确定出对方法的执行路径有特殊影响的参数值，将各种可能组合作为参数进行测试。

2. 面向对象的集成测试

传统的自顶向下和自底向上的集成策略对于面向对象的测试集成是没有意义的，类之间的相互依赖使其根本无法在编译不完全的程序上对类进行测试。因此，面向对象集成测试通常需要在整个程序编译完成后进行。此外，面向对象程序具有动态特性，程序的控制流往往无法确定，所以也只能对整个编译后的程序做基于黑盒的集成测试。

面向对象的集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。单元测试可以保证成员函数行为的正确性，集成测试则只关注于系统的结构和内部的相互作用。

面向对象集成测试可以分成两步进行：先进行静态测试，再进行动态测试。

静态测试主要针对程序结构进行，检测程序结构是否符合设计要求。现在常用的一些测试软件都能提供一种称为“可逆性工程”的功能，即通过源程序得到类关系图和函数功能调用关系图。将“可逆性工程”得到的结果与面向对象设计(OOD)的结果相比较，以检测面向对象编码(OOP)是否达到了设计要求。

动态测试则测试与每个动态语境有关的消息。设计测试用例时，通常需要上述的功能调用关系图、类关系图或实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，使得执行的测试能够达到一定覆盖标准。

3. 面向对象的系统测试

系统测试应该尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对暂时没有的系统设备部件，也应有相应的模拟手段。

在进行面向对象的系统测试时，应该参考面向对象分析(OOA)的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全“再现”问题空间。系统测试不仅是为了检测软件的整体行为表现，从另一方面看，也是对软件开发设计的再确认。

这里说的系统测试是对测试步骤的抽象描述，具体测试内容包括：

(1) 功能测试。

(2) 强度测试。

(3) 性能测试。

(4) 安全测试。

(5) 恢复测试。

(6) 可用性测试。

(7) 安装/卸载测试(install/uninstall test)等。

8.3 面向对象软件测试模型

面向对象的程序结构不再是传统的功能模块结构，作为一个整体，原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已变得不可行。而且，面向对象软件抛弃

了传统的开发模式，对每个开发阶段都有不同以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此，传统的测试模型对面向对象软件已经不再适用。

面向对象的软件开发模型将开发过程定义为面向对象分析（OOA）、面向对象设计（OOD）和面向对象编程（OOP）三个阶段。针对这种开发模型，应该建立一种新的测试模型。

在图 8-1 中，OOA Test 是指面向对象分析的测试；OOD Test 是指面向对象设计的测试；OOP Test 是指面向对象编程的测试；OO UNIT Test 是指面向对象单元测试；OO Integrate Test 是指面向对象集成测试；OO System Test 是指面向对象系统测试。

OO System Test
OO Integrate Test
OO UNIT Test
OOA Test
OOD Test
OOP Test
OOA
OOD
OOP

图 8-1　面向对象测试模型

8.3.1　面向对象分析的测试

面向对象分析（OOA）直接映射问题空间，全面地将问题空间中实现功能的实例抽象为对象，用对象的结构反映问题空间的复杂实例和复杂关系，用属性和服务表示实例的特性和行为。OOA 中认定的对象是指对问题空间中的结构、其他系统、设备、被记忆的事件、系统涉及的人员等实际实例的抽象，OOA 的测试重点在于测试其完整性和冗余性。OOA 阶段的测试划分为五个方面。

1. 对认定的对象的测试

对认定的对象进行测试时，主要测试五个方面：

第一，测试认定的对象是否全面，是否问题空间中所有涉及的实例都反映在认定的抽象对象中。

第二，测试认定的对象是否具有多个属性，只有一个属性的对象通常应看成其他对象的属性，而不是抽象为独立的对象。

第三，测试被认定为同一对象的实例是否有区别于其他实例的共同属性。

第四，测试被认定为同一对象的实例是否提供或需要相同的服务，如果服务随着不同的实例而变化，认定的对象就需要分解或利用继承性来分类表示。如果系统没有必要始终保持对象代表的实例的信息，提供或者得到关于它的服务，认定的对象也无必要。

第五，认定的对象的名称应该尽量准确、适用。

2. 对认定的结构的测试

认定的结构指的是多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系。认定的结构分为两种：分类结构和组装结构。分类结构体现了问题空间中实例的一般与特殊 的关系，组装结构体现了问题空间中实例整体与局部的关系。

对认定的分类结构的测试可从以下方面进行：

第一，对于结构中的对象，尤其是处于高层的对象，是否在问题空间中含有不同于下一层对象的特殊可能性，即是否能派生出下一层对象。

第二，对于结构中的对象，尤其是处于同一低层的对象，是否能抽象出在现实中有意义的更一般的上层对象。

第三，对所有认定的对象，是否能在问题空间内向上层抽象出在现实中有意义的对象。

第四，高层的对象的特性是否完全体现下层的共性。

第五，低层的对象是否有高层特性基础上的特殊性。

对认定的组装结构的测试则可以从以下方面进行：

第一，整体(对象)和部件(对象)的组装关系是否符合现实的关系。

第二，整体(对象)的部件(对象)是否在考虑的问题空间中有实际应用。

第三，整体(对象)中是否遗漏了反映在问题空间中有用的部件(对象)。

第四，部件(对象)是否能够在问题空间中组装新的有现实意义的整体(对象)。

3. 对认定的主题的测试

主题是在对象和结构的基础上更高一层的抽象，是为了提供 OOA 分析结果的可见性，如同文章中的各部分内容概要。对主题层的测试应该考虑以下方面：

第一，贯彻 George Miller 的“7 + 2”原则，如果主题个数超过 7 个，就要求对有较密切属性和服务的主题进行归并。

第二，主题所反映的一组对象和结构是否具有相同和相近的属性和服务。

第三，认定的主题是否是对象和结构更高层的抽象，是否便于理解 OOA 结果的概貌(尤其是对非技术人员的读者)。

第四，主题间的消息联系(抽象)是否代表了主题所反映的对象和结构之间的所有关联。

4. 对定义的属性和实例关联的测试

属性是用来描述对象或结构所反映的实例的特性。而实例关联是反映实例集合间的映射关系。对属性和实例关联的测试从如下方面考虑：

第一，定义的属性是否对相应的对象和分类结构的每个现实实例都适用。

第二，定义的属性在现实世界是否与这种实例关系密切。

第三，定义的属性在问题空间是否与这种实例关系密切。

第四，定义的属性是否能够不依赖于其他属性被独立理解。

第五，定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性体现。

第六，在问题空间中每个对象的属性是否定义完整。

第七，定义的实例关联是否符合现实。

第八，在问题空间中实例关联是否定义完整，特别需要注意一对多和多对多的实例关联。

5. 对定义的服务和消息关联的测试

定义的服务，就是定义的每一种对象和结构在问题空间所要求的行为。由于问题空间中 实例间必要的通信，在 OOA 中相应地需要定义消息关联。对定义的服务和消息关联的测试 从以下方面进行：

第一，对象和结构在问题空间的不同状态是否定义了相应的服务。

第二，对象或结构所需要的服务是否都定义了相应的消息关联。

第三，定义的消息关联所指引的服务提供是否正确。

第四，沿着消息关联执行的线程是否合理，是否符合现实过程。

第五，定义的服务是否重复，是否定义了能够得到的服务。

8.3.2　面向对象设计的测试

面向对象设计(OOD)是 OOA 的进一步细化和扩充，重点在于说明项目的实施方案，主要来确定类和类的结构。实施针对 OOD 的测试时，应针对功能的实现和重用以及对 OOA 结果的拓展进行，主要从以下三个方面考虑。

1. 对认定的类的测试

所测试内容应包括：是否涵盖了 OOA 中所有认定的对象；是否能体现 OOA 中定义的属性；是否能实现 OOA 中定义的服务；是否对应着一个含义明确的数据抽象；是否尽可能少地依赖其他类；类中的方法是否实现单个功能。

2. 对构造的类层次结构的测试

类层次结构是否涵盖了所有定义的类；是否能体现 OOA 中定义的实例关联；是否能实现 OOA 中定义的消息关联；子类是否具有父类没有的新特性；子类间的共同特性是否完全在父类中得以体现。

3. 对类库支持的测试

对类库的支持虽然也属于类层次结构的组织问题，但其强调的重点是再次软件开发的重用。由于它并不直接影响当前软件的开发和功能实现，因此，将其单独提出来测试，也可作为对高质量类层次结构的评估。测试时关注三个方面：首先，一组子类中关于某种含义相同或基本相同的操作，是否有相同的接口(包括名字和参数表)；其次，类中方法功能是否较单纯，相应的代码行是否较少；再次，类的层次结构是否是深度大，宽度小。

8.3.3 面向对象编程的测试

面向对象编码(OOP)就是编码实现 OOD 中的设计,实现软件的功能:面向对象程序把功能的实现分布在类中,类通过消息传递来协同实现设计要求的功能。这种程序架构能将出现的错误精确地确定在某一具体的类。在对 OOP 测试时,重点应集中在类功能的实现和相应的面向对象程序架构,主要体现为以下两个方面:

- 数据成员是否满足数据封装的要求。基本原则是检查数据成员是否被外界(数据成员所属的类或子类以外的调用)直接调用。
- 类是否实现了要求的功能。测试类的功能,不能仅满足于代码能无错运行或被测试的类所提供的功能无错误,应以所做的 OOD 结果为依据,检测类提供的功能是否满足了设计的要求,是否有缺陷。

总地来说,OOA Test 和 OOD Test 是对分析结果和设计结果的测试,主要对分析设计产生的文档进行测试,是软件开发前期的关键性测试。OOP Test 主要针对编程风格和程序代码实现进行测试,主要的测试内容在面向对象单元测试和面向对象集成测试中体现。

8.4 本章小结

面向对象开发技术的出现,对软件测试带来了前所未有的冲击。面向对象独有的特点,使得传统的测试技术不再适用。面向对象概念中所具有的全新特征,如封装、继承、多态等使得面向对象的软件开发更利于软件的复用,从而缩短了软件开发周期、提高软件开发质量,同时也能方便软件的维护。然而不可否认的是,与传统的开发手段相比,面向对象的开发方法增加了测试的复杂性,使得两者的测试方法和测试过程有了很大的不同。本章重点讨论了面向对象软件测试的不同层次及其特点,以及面向对象软件测试的模型。

习题 8

1. 测试面向对象软件和传统软件有何不同?
2. 什么是测试视角?从测试视角如何看待面向对象的基本概念?
3. 面向对象软件的测试模型是什么?
4. 面向对象软件测试的层次是怎样的?

第9章 软件测试自动化

软件测试是一项艰苦的工作,需要投入大量的时间和精力,据统计,软件测试会占用整个开发时间的40%。一些可靠性要求非常高的软件,测试时间甚至占到总开发时间的60%。但是软件测试具有一定的重复性,软件在发布之前要进行几轮测试。在测试后期所进行的回归测试中大部分测试工作是重复的,回归测试就是要验证已经实现的大部分功能,这种情况下,代码修改很少,针对代码变化所做的测试相对较少。而为了覆盖代码改动所造成的影响需要进行大量的测试,虽然这种测试找到软件缺陷的可能性小,效率比较低,但又是必要的。此后,软件不断升级,所要做的测试重复度也很高,所有这些因素驱动着软件测试自动化的产生和发展。

本章主要介绍软件测试自动化的含义、优点、局限性,软件测试自动化的引入条件和实施过程,以及软件测试工具。

9.1 软件测试自动化基础

软件测试自动化是相对于手工测试而言的。手工测试是靠测试人员的一步一步操作来完成的,测试人员操作一步,测试前进一步;测试人员停下来,测试也停下来。而测试自动化是通过测试工具来实现的,一旦测试启动,无论测试人员在做什么事情,测试都会继续下去;即使测试人员下班回家了,测试还可以在半夜自动启动、执行,并自动提交测试结果。自动化测试的目的是减轻手工测试的工作量,以达到节约资源,保证软件质量,缩短测试周期的效果,是软件测试中提高测试效率、覆盖率和可靠性的重要测试手段。也可以说,软件测试自动化是软件测试不可分割的一部分。

9.1.1 软件测试自动化的含义

什么是软件测试自动化?

根据软件质量工程协会关于软件测试自动化的定义:软件测试自动化就是利用策略、工具等,减少人工介入非技术性(unskilled)、重复性(repetitive)、冗长(redundant)的测试活动。

实际上,软件测试自动化通过执行用某种程序设计语言编制的自动测试程序,控制被测试软件的执行,来模拟手动测试步骤,分为全自动或是半自动测试。

全自动测试就是指在自动测试过程中,根本不需要人工干预,由程序自动完成测试的全

过程。半自动测试则在自动测试过程中，需要由人工输入测试用例或选择测试路径，再由自动测试程序按照人工制定的要求完成自动测试。

9.1.2 手工测试和自动化测试的比较

一言以蔽之，手工测试可以发现新缺陷，而自动化测试主要用于发现旧缺陷。软件测试可以采用手工测试和自动测试相结合的方法。在传统的测试方法中，测试工程师根据测试大纲中所描述的测试步骤和方法，手工输入测试数据，记录测试结果。手工测试的特点是能详细地测试软件的各个功能，测试速度由人来控制，能够完整而从容地观察软件的运行情况并立即报告测试结果。这当然是极好的。

但对于那些步骤与方法相对固定的测试可以采用自动测试方法。自动测试可以大规模地提高测试效率，减少测试工作量，具有可重复性，可以精确再现以前的测试步骤，有利于进行回归测试，可以降低人为的操作失误和对测试工程师的技术要求，从而降低测试成本，大大节约软件产品整个开发周期的费用，提高软件的质量。

9.1.3 软件测试自动化的局限性

自动化测试好处虽然很多，但并不是万能的，也存在着一定的局限性。

1. 软件自动测试并不能代替人的工作，尤其是带有智力性质的手工测试

工具本身不具有想象力，不能像手工测试一样进行发挥，不要期望将所有的测试活动进行自动化。软件测试工具不能发现所有的问题，测试工程师还需要做大量的工作。

2. 软件测试自动化可能降低测试的效率

当测试工程师只需要进行很少量的测试，而且这种测试在以后的重用性很低时，花大量的精力和时间去进行自动化的结果往往是得不偿失的。因为自动化的收益一般要在多次重复使用中才能体现出来。

3. 自动测试并非像测试工程师所期望的那样能发现大量的错误

测试首次运行时，可能发现大量错误。当进行过多次测试后，发现错误的几率会相对比较小，除非对软件进行了修改或在不同的环境下运行。事实证明新错误越多，自动化测试失败的几率就越大。发现更多的新缺陷应该是手工测试的主要目的。测试专家 James Bach 经过总结得出结论，85%的错误靠手工发现，而自动化测试只能发现 15%的错误。

4. 缺乏测试经验

如果测试的组织差、文档较少或不一致，则自动测试的效果比较差。

5. 测试自动化不能提高测试的有效性和仿效性

无论是自动测试还是手工测试都不影响测试的有效性和仿效性。如果测试用例本身是失败的，那么无论测试自动化程度有多高，测试的结果也是毫无意义的。自动测试只对测试

的经济性和修改性有影响。

6. 技术问题、组织问题和脚本维护

毫无疑问，商用测试工具是软件产品，作为第三方的软件产品，不具备解决问题的能力和技术支持。同样的原因，测试工具和其他软件的互操作性也是一个严重的问题，技术环境变化如此之快，使测试工具很难跟得上。自动化测试的推行有很多阻力，如组织不重视，拒绝成立这样的测试团队。对于测试脚本的维护工作量也很大，是否值得维护等问题都必须考虑。因此，软件自动化测试应该有正确的认识：它并不能完全代替手工测试。不要期望有了自动化测试就能提高测试的质量。如果测试工程师缺少测试的技能，那么测试也可能会失败。

9.2　软件测试自动化的作用和优势

工欲善其事，必先利其器。使用测试工具的目的，就是要提高软件测试的效率和软件测试的质量。前面已经介绍过，最常见的软件测试分类是白盒测试与黑盒测试。一些研究报告指出，黑盒测试所找到的软件缺陷的数量与白盒测试找到的数量是差不多的，有些时候甚至比白盒测试所找到的问题还要严重。这是因为黑盒测试的方向是以测试的广度为主，所进行的测试范围与种类比白盒测试广，因为广度的关系，有时候所找到的问题及其影响范围也相对较大。进行白盒测试只是为了确定程序代码的运行是否正确，而黑盒测试就类似一个把关的角色，白盒测试就是前端作业，黑盒测试就是后端验证。对软件测试来说，善于使用测试工具可以提供许多好处。但是，对于给定的需求，测试人员必须评估在项目中实施自动化测试是否合适。通常，自动化测试（与手工测试相对比）有如下好处。

1. 产生可靠系统

测试工作的主要目标：一是找出缺陷，从而减少应用中的错误；二是确保系统的性能满足用户的期望。为了有效地支持这些目标，在开发生存周期的需求定义阶段，当开发和细化需求时就应着手测试工作。

使用自动化测试可改进所有的测试领域，包括测试程序开发、测试执行、测试结果分析、故障状况和报告生成等。它还支持所有的测试阶段，其中包括单元测试、集成测试、系统测试、验收测试与回归测试等。

软件测试如果只使用人工测试的话，所找到的软件缺陷在质与量上都是有限的。在开发生存周期的所有领域中，假定自动化测试工具和方法被正确地实施，并且遵循定义的测试过程，则自动化测试有助于建立可靠的系统。通过使用自动化测试获得的效果可归纳如下：

1）需求定义的改进

可靠且节省成本的软件测试应开始于需求阶段。如果需求是明确的，并且始终如一地以可测试格式描述测试人员需要的信息，那么需求则被看做是具备测试条件的或可测试的。目前，许多工具有助于生成可测试的需求，如一些工具使用面向语法的编辑程序，诸如Lotus之类形式的语言编写，而一些其他工具，可建立图形化的需求模式。

2）性能测试的改进

手工进行性能测量的方法属于劳动密集型工作。例如，在对某产品进行手工性能测试时，需一名测试人员手工执行测试，另一名测试人员坐在一旁用秒表计时，这样的测试极易出错，并且不能确保自动重复。目前，已有许多性能测试工具，这些工具可使测试人员自动完成系统性能测试，给出计时数目与图形，并查明系统的瓶颈与阈值。测试工程师不必坐在一边，手握着秒表，通过启动测试脚本即可自动获取性能统计数据，这样测试人员也可腾出手来干一些创造性的、有智力挑战性的测试工作。

过去，需要许多不同型号的计算机，以及各类人员一遍又一遍地执行大量的测试，以产生统计上有效的性能数值。新的自动性能测试工具则可使测试人员利用文件或表格读出数据程序，或使用工具生成数据程序来实现，不管信息包含 1 行数据还是 100 行数据。

新一代测试工具可实现无人值守地运行性能测试，因为可预先设置测试执行时间，而后脚本自动开始，无须任何人工干预。许多自动性能测试工具允许虚拟用户测试。在虚拟用户测试时，测试人员可仿真几十个、几百个或几千个执行各种测试脚本的用户。在性能测试中，可使用负载来预测性能，并使用经过控制与测量的负荷来测量响应时间，性能测试结果分析将有助于软件性能的调整。

3）负载/压力测试的改进

支持性能测试的测试工具也支持压力测试。两种测试的差别仅在于如何执行测试。压力测试是使客户机在大容量情况下运行，以查看应会用将在何时、何处中断。在执行压力测试时，系统将经受最大和最小的负载，以便查明系统是否会中断，在何处中断，并确定哪一部分首先中断，从而识别系统的薄弱环节。系统需求应定义这些阈值，并描述系统对过载的反应。

完全使用手工方法对应用进行充分的压力测试是一项耗资大、困难多、不准确且耗时长的工作，需要大量用户和工作站参与测试过程，并且各种资源之间的结合不一定和谐。采用自动化测试后，压力测试不再需要 10 个以上的测试人员来完成。压力测试自动化对各方都有好处。例如，某一大型项目有 20 名测试人员，在一个星期测试的最后日子，要求 20 名测试人员星期六全体加班，进行压力测试工作。这样每一个测试人员都要以很高的速度对系统进行操作，这将给系统造成一定的压力。每一个测试人员都在同一时间内执行系统的最复杂的功能。而采用自动压力测试工具，当进行压力测试时，测试人员可以向工具发出何时执行压力测试、运行哪一个测试以及模拟多少个用户这样的指令，所有这一切无须用户干预。这样，测试人员不需要额外的资源，自动化测试工具通过在有限数量的客户机和工作站上仿真许多用户与系统的交互作用，为压力测试提供了另一种高效的选择方案。

许多自动化测试工具包括负载仿真器，该仿真器可使测试人员同时模拟几百个或几千个使用目标应用程序的虚拟用户。测试脚本的运行可以无人照管。绝大多数工具会产生一个测试日志输出，该输出将列出测试结果。

4）高质量测量与测试最佳化

自动化测试将产生高质量测量并实现测试最佳化。的确，自动化测试过程本身是可测量和可重复的。手工测试时，第二次测试的操作步骤不可能完全重复第一次测试的操作步骤。因此，手工测试很难产生任何类型一致的质量测量。而采用自动化测试技术，测试过程则是可重复且可测量的。测试人员对测量的质量分析，支持了测试工作最佳化，但只是在测

试可重复情况下才能做到。如前所述,自动化可实现测试的可重复性。例如,在手工执行测试时,如果测试人员发现了某种错误,需要重新建立测试,有时就难以获得成功。采用自动化测试,脚本可被回放,并且测试将是可重复且可测量的。另外,自动化测试可以产生许多度量(通常生成测试日志)。

5）改进系统开发生存周期

目前推出的若干自动化测试工具已支持开发生存周期的每一阶段。例如,在需求定义阶段有一些工具可以帮助生成具备测试条件的需求,以便减少测试工作量和测试成本。同样,支持设计阶段的工具,如建模工具,可记录测试用例的需求。测试用例代表用户实施系统级的各种组合操作,这些测试用例具有确定的起点、确定的用户(可能是人员或外部系统)、一组不连续的步骤以及确定的出口标准。

编程阶段也需要测试工具,如代码检查、度量报告、代码插装、基于产品的测试程序生成器等。如果需求定义、软件设计和测试程序已经进行了适当的准备,那么应用程序开发将会更高效地进行。在这些条件下,测试执行必定会更顺利。这些众多不同的测试工具,以一种方式或其他方式服务于整个系统开发生存周期,利于产生可靠的系统。

6）增加软件信任度

测试是自动执行的,所以不存在执行过程中的疏忽和错误,完全取决于测试的设计质量。一旦通过了强有力的自动测试,软件的信任度自然会增加。

2. 改进测试的工作质量

使用自动化测试工具,可增加测试的深度与广度,改进测试的工作质量。其具体好处可归纳如下。

1）改进多平台兼容性测试

自动化测试可以使脚本重用,支持从一个平台(硬件配置)到另一个平台的测试。计算机硬件、网络版本以及操作系统的变更可能给现有配置造成意外的兼容问题。在向大批用户展示产品的某个新应用之前,执行自动化测试可提供一种简捷的方法,确保这些变更不会对当前的应用程序与操作环境造成不利的影响。

2）改进软件兼容性测试

推动多平台兼容性测试的原理,同样适用于软件配置测试。软件变更(如升级或新版本的施行)可给现有软件带来意外的兼容性问题。执行自动化测试脚本可提供一种简捷的方法,确保这些软件变更不会对当前的应用与操作环境造成不利影响。

3）改进普通测试执行

自动化测试工具将消除重复测试的单调乏味。进行普通重复性测试时,测试人员可能厌烦一遍又一遍地测试同样单调的步骤。例如,一位测试人员负责完成“2000 年问题”测试。他的测试脚本把几百个日期放在 50 个屏幕上,有各种循环日期,以及一些必须重复执行的内容。唯一的不同是,在某一个循环内,他加上包含该日期的数据;在另一个循环内,他删除该数据;在其他循环内,他进行更新操作。此外,系统日期被重新设定以适应高风险的“2000 年日期”问题。同样的步骤重复了一遍又一遍,执行这些普通重复性测试时很快就会使人疲惫。如果实现了自动化,因为测试脚本不会在意是否必须一遍一遍地执行相同的单调步骤,并且能自动确认结果,测试工作就变得简单多了。

4）更好地利用资源

将烦琐的任务自动化，可以提高准确性和测试人员的积极性，将测试技术人员解脱出来以投入更多精力设计更好的测试用例。有些测试不适合自动测试，仅适合手工测试，将可自动测试的测试自动化后，可以让测试人员专注于手工测试部分，提高手工测试的效率。自动化测试为在允许的进度内更快速完成复杂测试提供了机会。也就是说，测试的自动建立使一些测试很快完成，同时也释放了测试资源，使测试人员可以将其创造力和工作转向更加复杂的问题与事务。

5）执行手工测试无法完成的测试

软件系统与产品变得越来越复杂，有时手工测试不能支持全部所需的测试。目前许多类型的测试分析人工无法完成。例如，判定覆盖分析或复杂度度量收集。判定覆盖分析验证程序上的每一输入点和出口至少已经调用过一次，并且验证程序上的每一判定已经在所有可能的出口上被经过至少一次。复杂度是通过源代码对可能的路径分析得出的，它已成为 IEEE 可靠软件测量标准的一部分。对于任何大型应用，将需要花费很多时间来计算代码的复杂度。对需要大量用户的测试，很难找到足够多的测试人员同时进行测试，但是自动化测试却可以模拟同时有许多用户，从而达到测试的目的。此外，使用手工测试方法几乎不可能进行内存泄漏测试。

6）重现软件缺陷的能力

测试人员在进行手工测试期间发现的缺陷，有多少能够原封不动地重现？自动化测试解决了这种问题。采用自动化测试工具，建立测试所采取的步骤被记录和存储在测试脚本中，脚本将回放早先执行的完全相同的顺序。为了进一步简化内容，测试人员可能把故障告诉相应的开发人员，开发人员可修改回放脚本的选项，以便直接产生软件错误的事件顺序。

3. 提高测试工作效率

使用测试工具来进行测试，能够节省时间并加快测试工作进度是毋庸置疑的，这也是自动化测试的主要优点。在前面的章节中曾经介绍过回归测试的重要性，这样的测试所耗费的时间是相当惊人的。要进行类似的软件测试，就必须借助软件测试工具来缩减测试过程，例如使用自动化测试软件来进行回归测试，就是一个很好的方式。有时使用自动化测试工具，测试人员并不能马上就体会到测试工作量的即刻或大量减少。以某些方式使用自动化测试工具，最初人们甚至会看到测试工作量增多的现象，这是因为需要完成一些建立任务。尽管测试工作量一开始可能增多，但在自动化测试工具实施第一次重复之后，测试工具投资回报将显现出来，因为人员的生产率提高了。

研究表明，使用自动化测试的总测试工作的人/小时数值仅占使用手工方法的 25%。

测试工作量的减少对测试施行期间的项目进度的加快可能影响最大。这一阶段的活动一般有测试执行、测试结果分析、缺陷纠正以及测试报告等。

1）测试计划制定——测试工作量增多

在做出引入自动化测试工具的决定之前，必须考虑测试过程的方方面面。应该对计划中的被测应用需求进行评审，以确定被测的应用是否与测试工具兼容。需要确认支持自动化测试的抽样数据的可用性，应该略述所需数据的类别与变量，制定获取或开发样本数据的

计划。关于要重用的脚本,必须定义和遵循测试设计与开发标准。必须考虑模块化与测试脚本的重用。因此自动化测试本身也需要开发工作,也具有自己的小型开发生存周期,这样将使测试计划工作量有所增加。

2）测试程序开发——测试工作量减少

测试程序的开发是一个缓慢、耗资且劳动密集的过程。当软件需求或软件模块改变时,测试人员常常不得不重新开发现有的测试程序,并从头开始生成新的测试程序。自动化测试工具允许使用图标单击选择和执行特定的测试程序。使用自动化测试相对于手工测试方法,测试过程、生成与修订时间会大大缩短,一些测试程序生成与修订工作甚至只需几秒钟的时间。使用测试数据生成工具,可减轻测试工作量。

3）测试执行——测试工作量减少/进度加快

测试执行的手工实现是劳动密集型的,容易出错误。测试工具允许测试脚本在执行期间回放,可减少人工干预。如果进行适当的设置,测试人员简单地启动脚本即可,由工具自动去执行,实现无人照管。必要时测试可进行多次,并且可在规定的时间开始,甚至通宵运行。这种无人照管的回放能力可使测试人员集中于其他工作。

4）对程序的回归测试——更方便/进度加快

这可能是自动化测试最主要的任务,特别在程序修改比较频繁时,效果是非常明显的。由于回归测试的动作和用例是完全设计好的,测试期望的结果也是完全可以预料的。将回归测试自动运行,可以极大地提高测试效率,缩短回归测试时间。由于测试是自动执行的,每次测试的结果和执行的内容的一致性可以得到保障,从而达到测试可重复的效果。

5）测试结果分析——测试工作量减少/进度加快

自动化测试工具一般包括一些种类的测试结果报告日志,并能维护测试记录信息。某些工具会按不同颜色输出结果,例如,绿色输出表示测试合格,红色输出表示测试不合格等,绝大多数工具可判别测试合格或不合格。这种测试记录输出提高了测试分析的简便性。绝大多数工具还允许故障数据与原始数据的对照,并自动指出两者的差别。

6）错误状态/纠正监视——测试工作量减少/进度加快

目前,一些自动化测试工具允许测试脚本发现故障后自动记录故障,只需很少的人工干预。以这种方式记录下的信息可以包含产生缺陷/错误的脚本的标识、正在运行的测试周期标识、缺陷/错误描述,以及出现错误的日期/时间等。例如,工具 TestStudio。只需简单地选择生成一个错误选项,只要脚本检测出有错误便生成错误报告,之后便可自动且动态地将缺陷连接到测试需求上,从而简化度量收集。

7）报告生成——测试工作量减少/进度加快

许多自动化测试工具内置有报告编写程序,以便用户生成和定制具体报告。即使那些没有内部报告编写程序的测试工具,也允许相关数据以所需的格式输入或输出。这样,将测试工具输出数据与支持报告生成的数据库集成便成了一件简单的工作。

软件自动化测试是软件测试技术的一个重要组成部分,引入自动化测试可以提高软件质量,节省经费,缩短产品发布周期。自动化测试可以进行基于功能、路径、数据流或控制流的覆盖测试,许多工作是手工测试所无法完成的。测试自动化如果实施正确的话,可以减小测试工作规模、加快测试进度、增强测试过程从而生产出可靠的产品。

9.3 软件测试自动化的引入

了解了软件测试自动化的基本概念和它的优缺点，我们就要考虑什么条件下适合引入自动化测试。首先，测试人员直至整个软件组织要对软件测试自动化有一个客观正确的认识，充分掌握自动化测试和手工测试的各自特点和应用范围，清楚软件测试自动化绝不能代替手工测试，而是两者相互补充。其次，将自动化测试融入整个开发过程，建立适合自动化测试的流程，并选择合适的测试工具，建立相应的测试环境和对测试人员进行充分的培训。最后，关注自动化测试的投入和产出，采取正确的测试策略，以确保从自动化测试中获得最佳效益。

9.3.1 软件测试自动化的正确认识

自动化测试能大大降低手工测试工作，但决不能完全取代手工测试。完全的自动化测试只是一个理论上的目标，实际上想要达到 100% 的自动化测试，不仅代价相当昂贵，而且操作上也是几乎不可能实现。一般来说，一个 40%～60% 的利用自动化的程度已经是非常好的了，达到这个级别以上将过大地增加测试相关的维护成本。

自动化测试能快速定位测试软件各版本中的功能与性能缺陷，但不会有创造性，不会发现测试脚本里没有设计的缺陷。测试工具不是人脑，测试设计者应将测试中各种分支路径的校验点进行定制；若定制不完整，即便事实上出错的地方，测试工具也不会发觉。因此，制订全面、系统的测试设计工作是相当重要的。

自动化测试能提高测试效率，但对于周期短、时间紧迫的项目不宜采用自动化测试。推行自动化测试的前期工作相当庞大，将企业级自动化测试框架应用到一个项目中也要评估其合适性，因此决不能盲目地应用到任何一个测试项目中，尤其不适合周期短的项目，因为很可能需要大量的测试框架的准备和实施而将项目拖垮。

实施测试自动化必须进行多方面的培训，包括测试流程、缺陷管理、人员安排、测试工具使用等。如果测试过程是不合理的，引入自动化测试只会给软件组织或者项目团队带来更大的混乱；如果我们允许组织或者项目团队在没有关于应该如何做的任何知识的情况下实施自动化测试，那将肯定会以失败告终。

9.3.2 对企业自身现状的评估分析

1. 从企业规模上来说，没有严格限制

无论公司大小，都需要提高测试效率，希望测试工作标准化、测试流程正规化、测试代码重用化。所以第一要做到的，就是企业从高层开始，直到测试部门的任何一个普通工程师，都要树立实施自动化测试的坚定决心，不能抱着试试看的态度。一般来说，一个这样的软件开发团队可以优先开展自动化测试工作，测试开发人员比例合适，比如 1∶1～1∶1.5；开发团队总人数不少于 10 个。当然，如果你的公司只有三五个测试人员，要实施自动化测试绝非易事；不过可以先让一个、两个测试带头人首先试着开展这个工作，不断总结、不断提高，

并和上司经常汇报工作的开展情况，再最终决定是否全面推行此事。

2. 从公司的产品特征来说，一般开发产品的公司实施自动化测试要比开发项目的公司条件优越些

原因很简单，就是测试维护成本和风险都小。产品软件开发周期长，需求相对稳定，测试人员有比较充裕的时间去设计测试方案和开发测试脚本；而项目软件面向单客户，需求难以一次性统一，变更频繁，对开发、维护测试脚本危害很大，出现问题时一般都以开发代码为主，很难照顾到测试代码。但绝不是说做项目软件的公司就不能实施自动化测试。当前国内做项目的软件公司居多，只要软件的开发流程、测试流程、缺陷管理流程规范了，推行自动化测试自然水到渠成。

3. 标准化的开发和管理流程

（1）把你想做的写下来（计划管理）；
（2）按照你写下来的去做（行为管理）；
（3）把做的事情记录下来（报告管理）；
（4）出现的问题要设法解决（跟踪管理）。

在测试流程里，这几个要点都一一有所落实；如果你的软件开发团队据此开发软件，那么完全具备实施自动化测试的条件。

9.3.3 软件测试自动化的引入条件

测试工具本身的优势并不意味着使用测试工具就能成功，关键还在于使用工具的人。很多刚拥有测试工具的人，经常过分夸大工具的功效，并投入太高的期望。但是，工具只是提供了解决问题的一种手段而已，成功的测试自动化需要下面一些关键的因素：

1. 管理层要充分意识到软件测试自动化的重要性

由于软件测试自动化在前期的投入要比手工测试的投入大得多，除了在购买软件测试工具之外，还要进行大量的人员培训。因此，管理层对软件测试自动化有一个正确的认识，是非常重要的，若管理层对此持漠视态度，有效地开展软件测试自动化几乎是不可能的。

2. 有个很好的计划和稳定的应用行为

凡事预则立，不预则废。一个软件团队实施测试自动化，绝不是拍脑袋说干就能干好的，它不仅涉及测试工作本身流程上、组织结构上的调整与改进，还包括需求、设计、开发、维护及配置管理等其他方面的配合。测试自动化对于那些没有很好定义的应用，是无法进行的。即使是一个相当稳定的应用，如果测试人员不了解它的行为和相关特定领域内的问题和需要，那么测试自动化也会充满问题。为了成功地进行测试自动化，测试人员需要理解并预知应用的行为，然后才能按测试计划使用测试工具。只有在计划适当、测试工具合适、自动化测试过程定义明确的情况下，自动化测试所需的总测试工作量才能减少，并得到较高的测试质量。

3．实施测试自动化必须进行多方面的培训

实施测试自动化必须进行多方面的培训，包括测试流程、缺陷管理、人员安排、测试工具使用等。

4．一个专注的、有着丰富技能的测试团队，并且被分配了足够的时间和资源

软件开发是团队工作，在这一领域要尤其注重以人为本，所以人员之间的配合、测试组织结构的设置非常重要，每个角色一定要将自己的责任完全担负起来。测试工具会依据所进行的软件测试项目的不同而不同，但是其使用的目的是基本相同的。测试人员平时就应该学习使用并搜集测试工具，因为要想在软件测试时充分发挥测试工具的功效，就必须非常熟悉如何使用工具，以及使用工具的适当时机，一位优秀的测试人员会不断地进修专业知识。参加测试的人员必须熟悉正在测试的应用，并且必须具有有关平台与网络以及所使用的自动化测试工具方面的特定技能。

由于测试件是软件，测试自动化应当像其他独立的项目支持管理一样分配资源和时间，否则最终将会导致失败。测试人员应在生存周期初期参与需求与设计评审，了解业务需求，提高需求可测试性并支持有效的测试设计与开发。在使用自动化测试工具时，这是一种必不可少的重要活动。

综上所述，一般这样的测试条件下使用自动化测试：

(1) 项目没有严格的时间压力。

(2) 具有良好定义的测试策略和测试计划(知道要测试什么，知道什么时候测试)。

(3) 对于自动化测试你拥有一个能够被识别的测试框架和候选者。

(4) 能够确保多个测试运行的构建策略。

(5) 多平台环境需要被测试。

(6) 拥有运行测试的硬件。

(7) 拥有关注在自动化过程上的资源。

(8) 被测试系统是可自动化测试的。

9.4 软件测试自动化的实施

9.4.1 软件测试自动化的流程框架

软件自动化测试工具标准流程提供了一套完整的测试流程框架，软件测试团队可以以它为基础，根据业务发展的实际要求，定制符合团队的软件测试流程。软件自动化测试标准流程如图9-1所示。

(1) 制定测试计划的目的是确定和描述要实施和执行的测试。这是通过生成包含测试需求和测试策略的测试计划来完成的。可以制定一个单一的测试计划，描述所有要实施和执行的不同测试类型，也可以为每种测试类型制定一个测试计划。

(2) 设计测试的目的是确定、描述和生成测试过程和测试用例。

(3) 实施测试的目的是实施(记录、生成或编写)设计测试中定义的测试过程。输出工

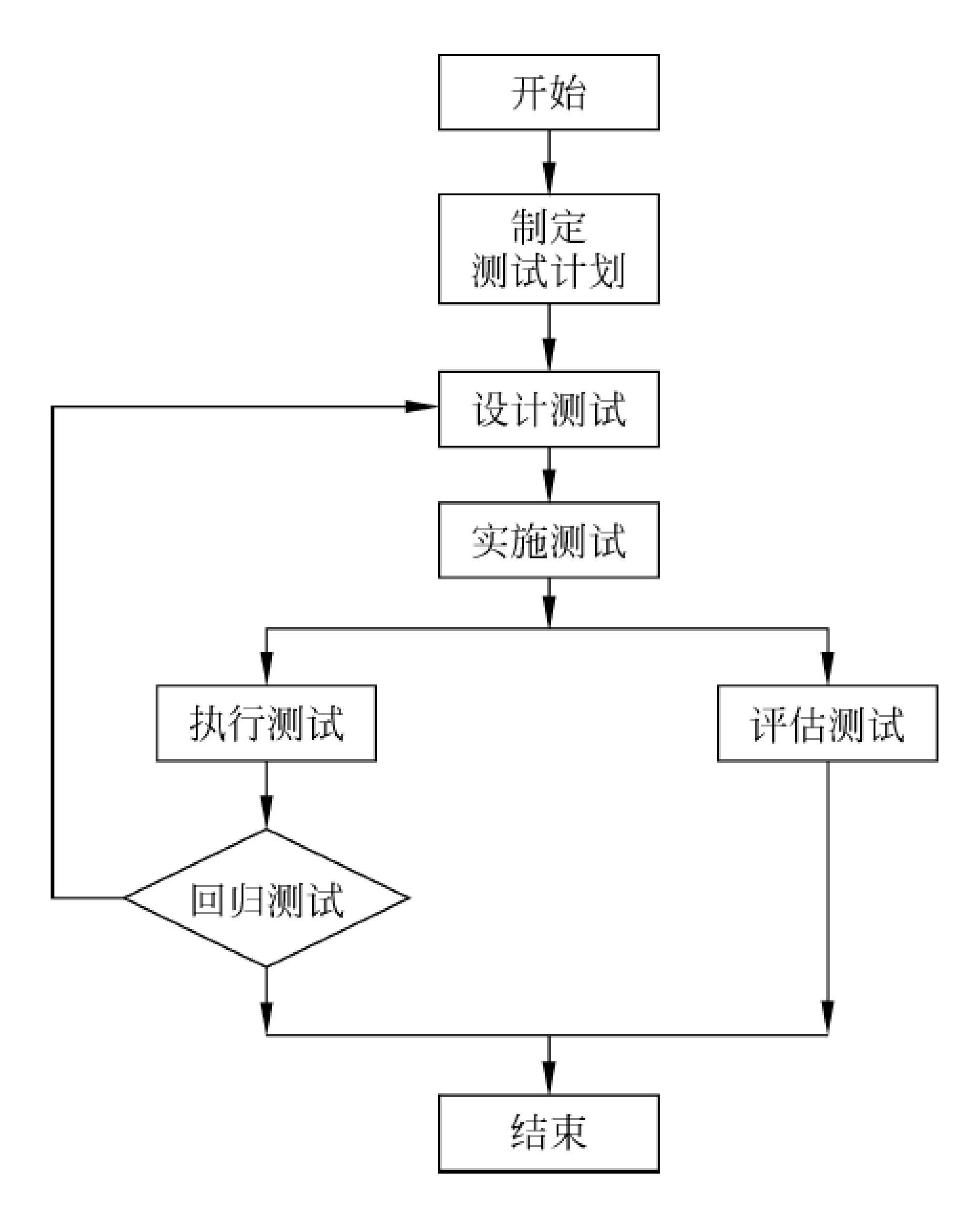


图 9-1　软件测试自动化流程图

件是测试过程的计算机可读版本，称为测试脚本。

（4）执行测试的目的是确保整个系统按既定意图运行。系统集成员在各迭代中编译并链接系统。每一迭代都需要测试增加的功能，并重复执行以前版本测试过的所有测试用例（回归测试）。

（5）评估测试的目的是生成并交付测试评估摘要。这是通过复审并评估测试结果、确定并记录变更请求，以及计算主要测试评测方法来完成的。测试评估摘要以组织有序的格式提供测试结果和主要测试评测方法，用于评估测试对象和测试流程的质量。

9.4.2　软件测试自动化的实施过程

为了实现软件测试自动化，首先要具备一套自动化测试工具软件。但是，并不是有了这个工具软件就能把测试自动化做好。为了做好自动化测试，需要经历计划、实施及不断完善这样一个过程。在这个过程中要做以下这些工作。

1. 熟悉、分析测试用例

首先应按各用例的描述对用例执行手工测试，至少全部实现一遍，直到对这些用例的每一步及其判断准则都有深入的了解。只有这样，在编写自动化测试程序时才可以正确模拟手工测试的整个过程，编写起来才能够得心应手。

2. 把已有的测试用例归类，写成比较简单的测试自动化计划书

可以按照软件的功能来分，如用户登记、查询等。除此之外，还可以按照网页（网络软件类）来划分。在具体做这项工作时，可以按计划来系统地进行。

3. 开始自动化测试程序的编写

由于测试自动化计划书中已经将测试用例分类，让测试人员负责不同的部分，平衡作业，这样可以节省时间。

在测试工作中，一般都会用测试工具记录的功能，按测试用例的步骤走一遍，然后再在由此产生的"记录"上进行编辑。各种自动测试工具有各自的特点，因而各自记录产生的结果都会不同，所以在这里也很难一概而论地描述哪些地方需要进行改动，否则重复执行的时候会出现问题。这个问题需要在实践中去探索，并找出结论。但是，所有的自动测试工具所生成的"记录"，都要按测试用例的不同进行编辑，这一点是肯定的。一般情况下，要加入的有说明(各步骤的目的)各变量的赋值与定性、各种循环结构语句、出错的判断语句及出错报告语句等。在编辑完毕后，还要对这个自动测试程序进行调试。

4. 尽量用"数据驱动"来提高测试覆盖率

通常，仅测试用户新建档案这一功能就有几十种不同的数据组合需要测试。采用手工测试要花很长时间，如果任务很紧，那会是一件头痛的事。做测试跟编程一样，经常需要赶时间，如果时间太紧迫，那么只能执行其中的一部分。这样测试覆盖率就大打折扣了。但是，如果将所有的不同数据组合都放到一个编辑文件里(如记事本)，各数据间用固定的符号或空格分开，每一组各占一行，这样就可以在原来编好的自动测试程序里加入数据驱动部分，让其在执行时将存有数据组合的文件一行一行地读进，从而完成所有的组合测试。而测试人员可以在一边观察，或去做其他的事情，根本不用管，只要等到测试结束后检查测试报告就可以，这是非常方便的。

5. 将测试用例写成自动化测试程序

在建立了自动化测试的框架以后，所要做的就是不断地输送新的自动化测试程序，直到所有写成自动化测试程序的测试用例(也就是用所有自动化测试设计书中列出的自动化测试用例来取代手工测试用例)都变成程序为止。

6. 不断地完善自动化测试系统

每当接到用户或是他人报告软件缺陷时，测试人员应马上按报告描述的情况手工测试一次，看是否能重现报告的问题。当缺陷被纠正后，应将这次手工测试自动化，用于日后的重复测试。这一类自动化测试的效率是很高的，因为在实际工作中，发现的缺陷虽被纠正，但常常会由于各种原因而又出现在软件中。最常见的原因之一就是软件编程人员在新版本中，忘记将已纠正的部分代替出错的部分放进将要编译的程序库中。

不断增加新的测试程序或对已有的测试程序进行修改。测试中的软件通常会遇到新增加一些功能的情况，这时，就要增加测试用例，并针对这些新增加的功能编写自动化测试程序。

同样，测试中的软件通常会因各种原因而做出一些修改，如功能方面的修改或网页排版上的修改等。在这种情况下，测试人员就要按照开发人员提供的情况，对相关的测试程序进行修改。

测试自动化是一个庞大的工程，因此在真正动手之前，必须尝试把所有的因素及可能性研

究一遍，然后制定方案。这一步是绝对不可以忽略的，因为此方案一定，日后的工作就要按规定去做。如果漏掉了一些重要的因素，以后才发现，那么对其改正就要付出代价，浪费许多时间和人力。因此，在制定方案的时候要反复推敲，尽可能把现有的和将来的因素都考虑在内。

9.5 软件测试工具分类

测试工具可以从以下两个不同的方面去分类。

根据测试方法不同，分为白盒测试工具和黑盒测试工具。

根据测试的对象和目的，分为单元测试工具、功能测试工具、负载测试工具、性能测试具和测试管理工具等。

1. 白盒测试工具

白盒测试工具针对程序代码、程序结构、对象属性、类层次等进行测试，测试中发现的缺陷可以定位到代码行、对象或变量级。根据测试工具原理的不同，又可以分为静态测试工具和动态测试工具。

静态测试工具可对代码进行语法扫描，找出不符合编码规范的地方，并根据某种质量模型评价代码的质量，生成系统的调用关系图等。它直接对代码进行分析，不需要运行代码，也不需要对代码编译链接、生成可执行文件。

动态测试工具与静态测试工具不同，需要实际运行被测系统，并设置断点，向代码生成的可执行文件中插入一些监测代码，从而掌握断点这一时刻程序运行数据（对象属性、变量的值等）。

2. 黑盒测试工具

黑盒测试工具适用于系统功能测试和性能测试，包括功能测试工具、负载测试工具、性能测试工具等。黑盒测试工具的一般原理是利用脚本的录制（Record）/回放（Playback）模拟用户的操作。然后将被测系统的输出记录下来，同预先给定的标准结果比较。黑盒测试工具可以大大减轻黑盒测试的工作，在迭代开发的过程中，能够很好地进行回归测试。

3. 其他测试工具

在上述两类测试工具之外还有测试管理工具，这类工具负责对测试计划、测试用例、测试实施进行管理，对产品缺陷跟踪管理、产品特性管理等。

此外，还有一些专用的测试工具，例如，针对数据库测试的 TestBytes、对应用性能进行优化的 EcoScope 等工具。

9.6 几种常用软件测试工具

9.6.1 性能测试工具 LoadRunner

LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具，可以模拟上千

万用户实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业架构进行测试。通过使用 LoadRunner，企业能最大限度地缩短测试时间、优化性能和加速应用系统的发布周期。

目前企业的网络应用环境都必须支持大量用户，网络体系架构中又含有各类应用环境且由不同供应商提供软件和硬件产品。难以预知的用户负载和愈来愈复杂的应用环境，使公司时时担心会发生用户响应速度过慢、系统崩溃等问题。这些都不可避免地导致公司收益的损失。Mercury Interactive 的 LoadRunner 能让企业保护自己的收入来源，无须购置额外硬件而最大限度地利用现有的 IT 资源，并确保终端用户在应用系统的各个环节中对其测试应用的质量、可靠性和可扩展性都有良好的评价。

LoadRunner 的工作过程如下：

首先，LoadRunner 的虚拟用户生成器（Virtual User Generator，简称 VuGen）引擎可以方便地创立系统负载，允许测试人员以虚拟用户的方式模拟真实的业务操作行为。利用虚拟用户，还可以在 Windows、UNIX 或 Linux 机器上同时产生多用户访问，所以 LoadRunner 能极大地减少负载测试所需的硬件和人力资源。

其次，LoadRunner 内含集成的实时监测器，在负载测试过程中，可以观察到应用系统的运行性能。包括交易性能数据（如响应时间）、系统组件（如 Application Server、Web Server）、网络设备以及数据库等的实时性能。在测试过程中，可从客户和服务器两方面评估这些系统组件的运行性能，从而更快地发现问题。

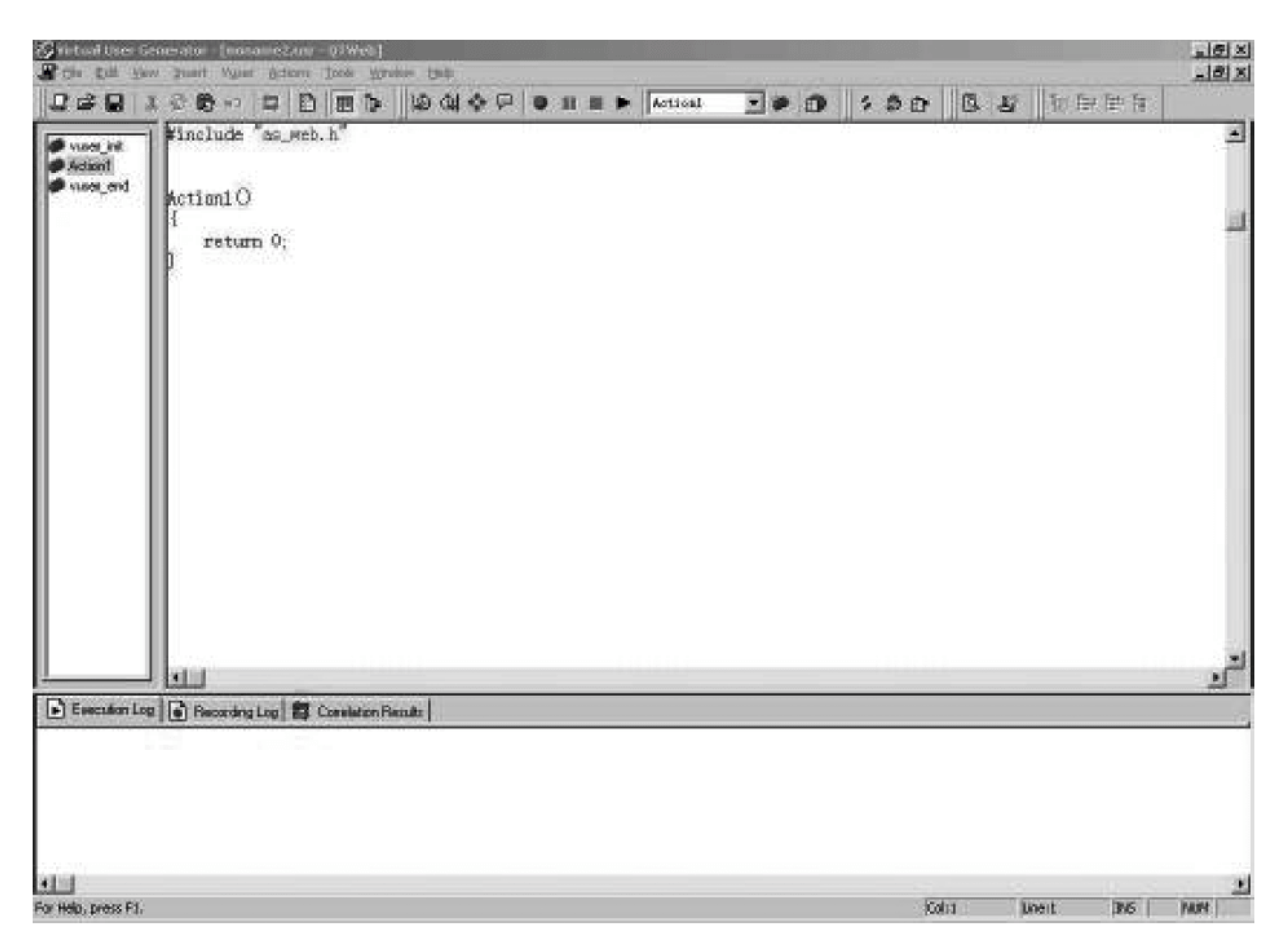


图 9-2　LoadRunner 主窗口

最后，测试完毕后，LoadRunner 收集汇总所有测试数据，并提供高级的分析和报告工具，以便快速发现性能问题。

整个工作过程可以总结为以下六个步骤：

（1）制定负载测试计划。

（2）开发测试脚本。

（3）创建运行场景。

（4）运行测试。

（5）监视场景。

（6）分析测试结果。

图 9-2 是 LoadRunner 主窗口。

9.6.2　功能测试工具 WinRunner

WinRunner 是 Mercury Interactive 公司的企业级的功能测试工具，可用于检测应用程序是否能够达到预期的功能及正常运行。通过自动录制、检测和回放用户的应用操作，WinRunner 能够有效地帮助测试人员对复杂的企业级应用的不同发布版进行测试，提高测试人员的工作效率和质量，确保跨平台的、复杂的企业级应用无故障发布及长期稳定运行。

WinRunner 的工作过程主要包括如下 6 个阶段：

（1）创建 GUI Map 文件：WinRunner 可以通过它来识别被测试应用程序中的 GUI 对象。

（2）创建测试脚本：通过录制、编程，或两者结合来创建。在录制测试脚本时，在你想检查被测试应用程序响应的地方插入验证点。

（3）调试脚本：用调试（Debug）模式运行测试脚本以确保它们可以平稳地运行。还可以使用 WinRunner 提供的 Step、Step Into、Step out 功能来调试脚本。

（4）运行测试：用验证（Verify）模式运行测试脚本来测试你的应用程序。当 WinRunner 在运行中碰到验证点时，它会将被测应用程序中的当前数据和以前捕捉的期望数据进行比较，如果发现了任何不匹配，WinRunner 将会把目前的情况捕捉下来作为真实的结果。

（5）检查结果：确定测试脚本的成功或是失败。在每次测试脚本运行结束之后，WinRunner 会将结果显示在报告中。它描述了所有在运行中碰到的重要的事件，例如验证点、错误信息、系统信息或是用户信息。如果发现在运行中有任何不匹配的验证点，你可以在测试结果窗口中查看期望的和实际的结果。

（6）提交缺陷：如果一个测试脚本是由于所测试应用程序中的缺陷而导致失败的，你可以直接从测试结果窗口中提取缺陷的相关信息。

图 9-3 是 WinRunner 欢迎界面。

图 9-4 是 WinRunner 脚本编辑界面。



图 9-3　WinRunner 欢迎界面

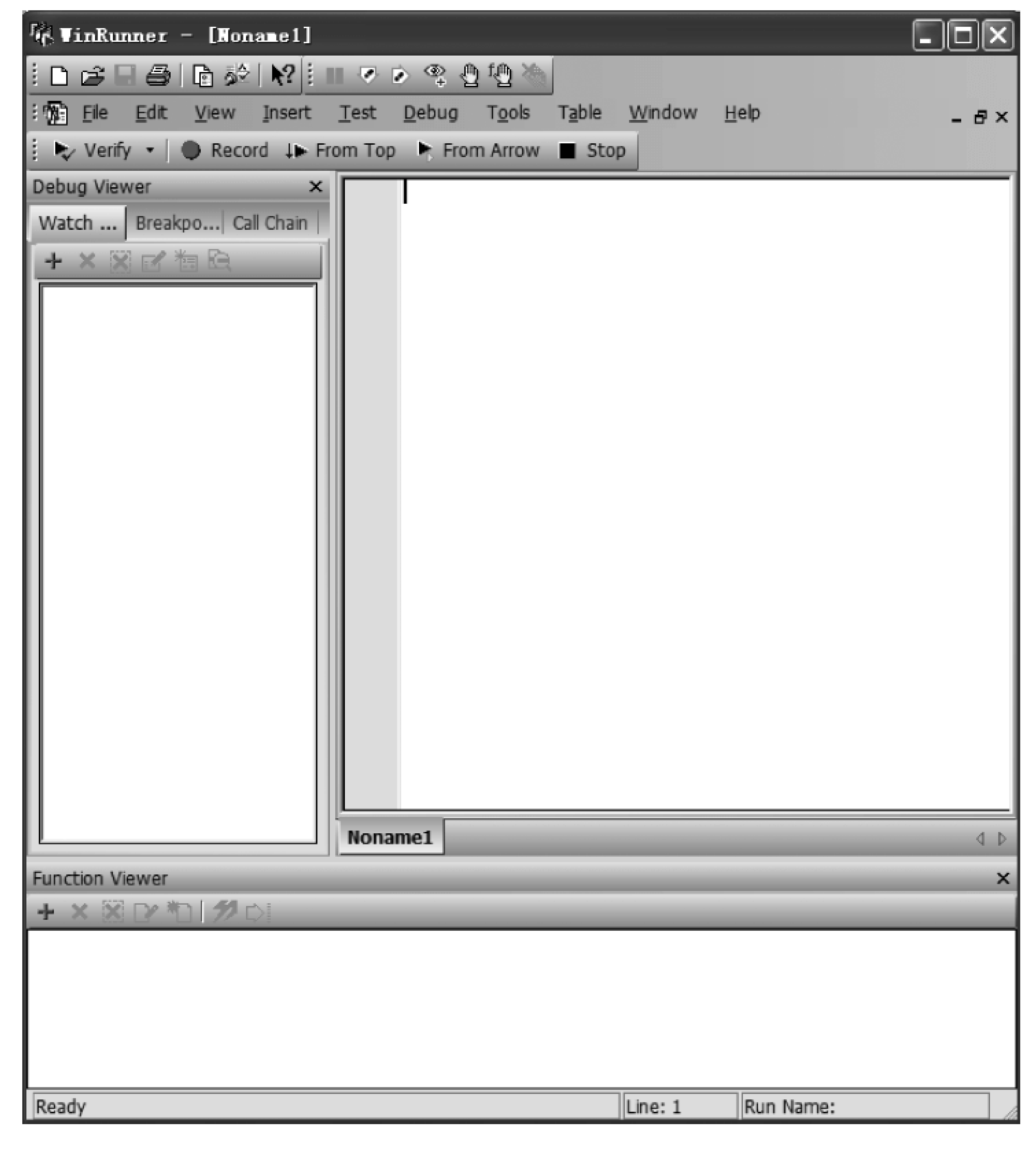


图 9-4　WinRunner 脚本编辑界面

9.6.3 白盒测试工具 JUnit

JUnit 是由《设计模式》的作者 Erich Gamma 和极限编程(eXtreme Programming,简称 XP)的创始人 Kent Beck 编写的一个回归测试框架(regression testing framework)。JUnit 其实是一套框架,只要继承 TestCase 类,就可以用 JUnit 进行自动测试了。JUnit 测试是程序员测试,即所谓白盒测试,因为程序员知道被测试的软件如何完成功能和完成什么样的功能。

在 JUnit 单元测试框架的设计时,设定了三个总体目标:第一个是简化测试的编写,这种简化包括测试框架的学习和实际测试单元的编写;第二个是使测试单元保持持久性;第三个则是可以利用既有的测试来编写相关的测试程序。

通过 JUnit 可以用 Mock Objects 进行隔离测试;用 Cactus 进行容器内测试;用 Ant 和 Maven 进行自动构建;在 Eclipse 内进行测试;对 Java 应用程序、Filter、Servlet、EJB、JSP、数据库应用程序、标签库等进行单元测试。

使用 JUnit 时,主要都是通过继承 TestCase 类来撰写测试用例,使用 test…()名称来撰写单元测试。

用 JUnit 进行单元测试需要做四件事:

(1) 用一个 import 语句引入所有 junit. framework. * 下的类。

(2) 使用 extends 语句继承 junit. framework. TestCase 类。

(3) 自行添加一个 main 方法调用 TestRunner. main (测试类名. class)。

(4) 用一个调用 super(string)的构造函数。

在阅读 JUnit 代码时,还会发现有许多以 test 开头的方法,而这些方法正是需要测试的方法,JUnti 测试其实只要在所有 test 开头的方法中对数据添加断言方法。

9.6.4 测试管理工具 TestDirector

TestDirector 是 HP Mercury 公司基于 Web 的测试管理工具,它在一个整体的应用系统中集成了测试管理的各个部分,包括需求管理、测试计划、测试执行以及错误跟踪等功能,从而有效地整合了测试过程。

TestDirector 采用集中式的项目信息管理,后台采用集中式的数据库(Oracle、SQL Server 或 Access 等)。所有的关于项目的信息都按照树状目录方式存储在管理数据库中,只有被赋予权限的用户才可以登录和查询、修改。

基于 Web 的测试管理系统提供了一个协同合作的环境和一个中央数据库。TestDirector 测试管理系统能让用户无论何时何地都能参与整个测试过程。从整体来看,TestDirector 可实现完全基于 Web 的用户访问,用户界面和访问权限可定制,实现完全基于 Web 的服务器管理、用户组和权限管理,实现测试管理软件的远程配置和控制。

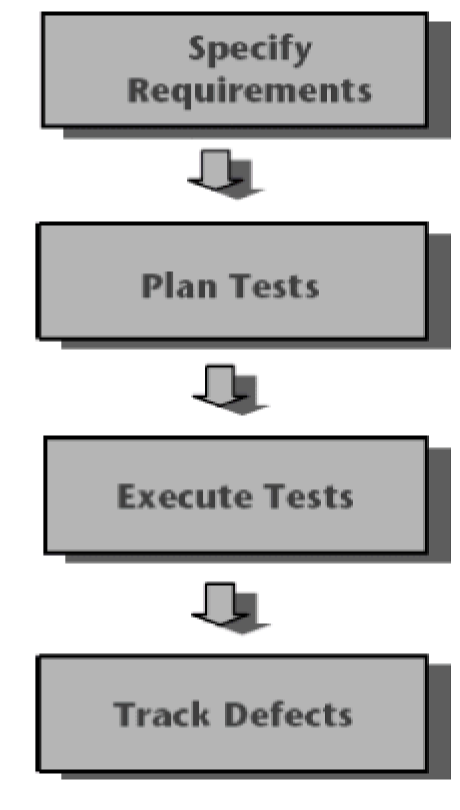


图 9-5 TestDirector 测试管理阶段图

TestDirector 的测试管理包括如下四个阶段,如图 9-5

所示。

- 需求定义(Specify Requirements):分析应用程序并确定测试需求。
- 测试计划(Plan Tests):基于测试需求,建立测试计划。
- 测试执行(Execute Tests):创建测试集(Test Set)并执行测试。
- 缺陷跟踪(Track Defects):报告程序中产生的缺陷并跟踪缺陷修复的全过程。

图 9-6 是 TestDirector 的主界面。



图 9-6 TestDirector 主界面

9.6.5 专用测试工具 WAST

Web Application Stress Tool (Web 应用负载测试工具,以下简称 WAST)是由微软公司开发的专门用来进行实际网站压力测试的一套工具,它支持身份验证、加密和 cookies,可用于模拟 Web 浏览器发送请求到任何采用了 HTTP 1.0/1.1 标准的服务器,而不考虑服务器运行的平台。它可使用少量的客户端计算机模拟大量用户上线对网站服务器可能造成的影响。网站实际上线前,先对其进行仿真环境下的测试,可以找出系统潜在的问题。

WAST 还有很多有用的特性,如对于需要署名登录的网站,它允许创建用户账号;允许为每个用户存储 cookies 和 Active Server Pages(ASP)的 session 信息;支持随机或顺序的数据集;支持带宽调节和随机延迟,以更真实地模拟现实情形;支持 WSecure Sockets Layer (SSL)协议;允许 URL 分组和提供对每组点击率的说明;提供一个对象模型,可以通过 VBScript 处理或通过定制编程来达到开启、结束和配置测试脚本的效果。

WAST 测试过程包括准备测试脚本、设置与调整测试脚本/测试策略、执行测试与分析

测试结果三个步骤。

9.7 本章小结

软件测试自动化是软件测试技术的一个重要组成部分，能够完成许多手工无法完成或者很难实现的测试工作。正确、合理地实施自动化测试，能够快速、全面地对软件进行测试，从而提高软件质量，节省经费，缩短产品发布周期。我们应该合理地进行自动化测试的安排和计划，合理运用自动化测试工具，使测试工作高效率完成。

习题 9

1. 什么是软件自动化测试？
2. 自动化测试的作用和优势有哪些？
3. 比较手工测试和自动化测试。
4. 简述软件测试自动化的实施过程。
5. 简述软件测试工具的分类。
6. 简述软件测试自动化的引入条件。
7. 有哪些常用的软件测试工具？

第10章 测试实践——一个实际软件项目的测试案例

本章将通过一个实际软件项目的测试案例，让读者加深对软件测试技术和软件测试过程的理解，使理论得以应用。

对于任何一个软件项目来说，都不能盲目地照搬其他软件项目的测试过程，应该根据被测项目来制定适合的测试方案，并不一定要经历所有的测试过程。但测试过程中应该多取他人之长，多看一些典型的测试实例。储备了一定的理论知识，并从实践中积累了自己的经验，测试就显得得心应手了。

本章介绍的被测试项目是B/S架构的网上书店管理系统。给读者详细讲解了测试计划、测试用例、缺陷报告、测试结果总结与分析等内容。测试用例将针对该网上书店管理系统的一个模块——用户注册、登录和注销模块。该模块不但包含了对数据库的应用，还对系统的安全性、准确性、高效性等有很高的要求。

10.1 被测试项目介绍

10.1.1 被测系统概述

本管理系统采用ASP＋Access的模式，系统包含了常见的网上购物系统应有的所有模块，如用户注册登录和管理模块、用户在线选书购书模块、销售管理模块等。具备完善的后台管理功能，具备用户管理、图书管理、新闻通知管理、网站信息管理等模块。系统还使用了登录验证码技术、MD5加密算法，增强系统的安全性。还有一些功能，如图片、广告上传后自动显示在页面，删除图书时自动删除相关图片，更新图片或网站广告时自动删除旧版，简单实用的图片缩略功能。

本系统可以作为中小型书店的电子商务网站，也可以方便地移植成其他的类似业务的电子商务管理系统。主要具备的功能如下。

1. 客户端功能部分

- 游客可以查看最新、分类、特价、排行榜等所有图书。
- 游客可以查看网站新闻。
- 游客可以查看图书详细信息。

- 游客可以给网站提交留言。
- 游客可以对图书提交评论。
- 用户注册、登录。
- 用户可以查看、修改个人信息。
- 用户可以找回密码。
- 用户可以使用购物车功能。
- 用户可以购买图书、下订单、查看、修改订单。

2. 网站后台管理部分

- 图书类别管理：增加、修改、删除图书大类和小类。
- 图书管理：添加、修改、删除图书。
- 用户管理：查看、修改、删除普通用户和管理员。
- 图书订单管理：查看订单、修改订单状态、删除订单。
- 信息管理：添加、修改、删除新闻和通知。
- 网站信息设置：设置、修改网站信息、网站广告、服务等。

图 10-1 所示是系统总体设计方案。图 10-2 是网上书店网站的首页。

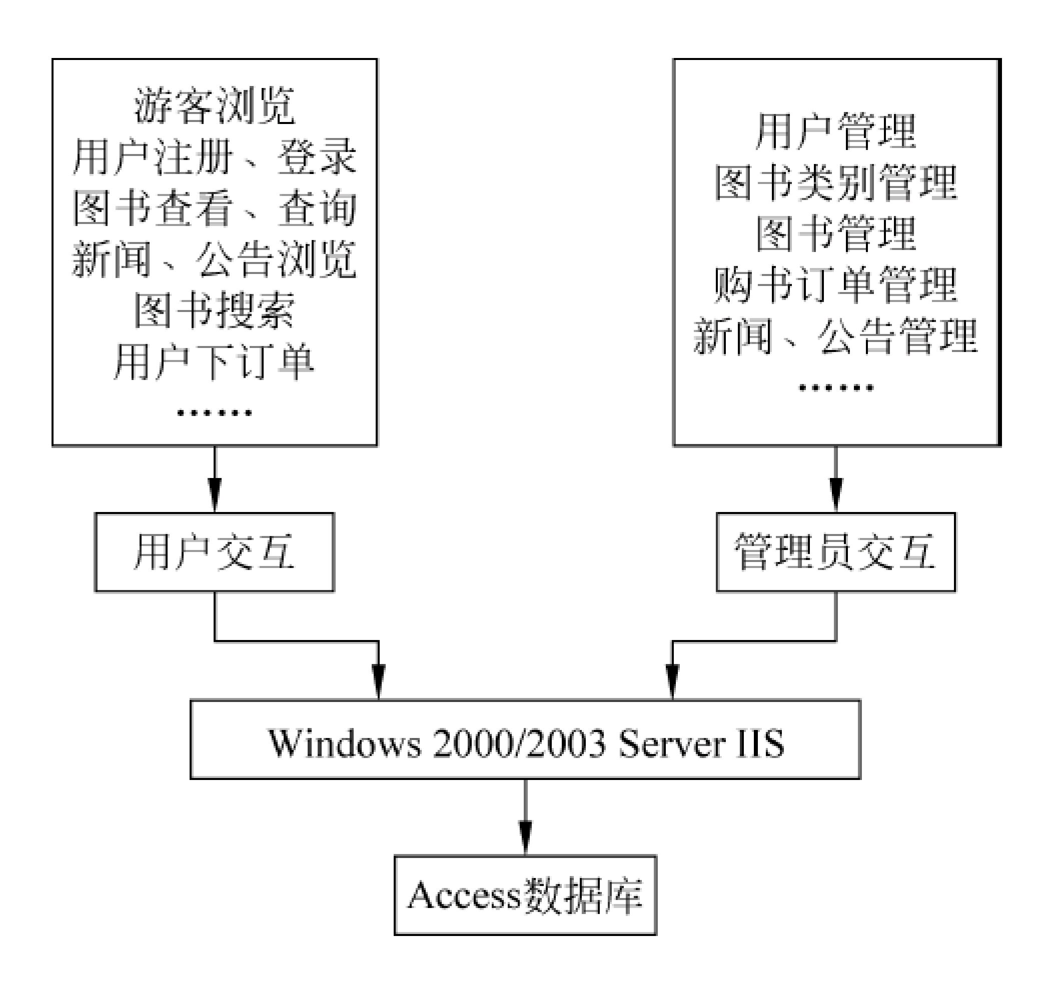


图 10-1 系统总体设计方案

图 10-3 是读者找到一本图书，查看详细信息的页面，从图中可以看到图书封面图是小图（缩略图），当单击"查看图书封面大图"后，就可以看到封面大图效果。图书定价、书店售价、图书介绍、目录等信息，一目了然。如果是注册用户，此时就可以进行"购买"或"放入购物车"操作了。

图 10-4 是一个注册用户登录后查看自己的订单的页面，可以看到用户操作界面简单明了，即使是新手也会很快操作自如。

图 10-5 是网站管理员登录后台管理系统后，正在进行图书查看、管理的页面。可以看出，左侧是管理页面的导航菜单，所有的管理操作都可以在这一个页面中完成。

高远网上书店

用户 密 码

验证 8150 登录 注册

设为首页
加入收藏
联系我们

今天是 2007年5月29日 星期二 书店导航 首 页 | 新书上架 | 分类图书 | 图书排行榜 | 特价图书 | 意见留言 | 本站新闻

用户服务中心： 注 册 | 我的购物车 | 我的订单 | 修改密码 | 忘记密码 | 我的资料 | 统计信息 | 注销登录

图书查找 请输入关键字 书 名 立即查询 高级查询 欢迎访问本站，祝您购书愉快！

最 新 新 闻

- 高远书店将于2007年…
- 祝贺高远网上书店，戒读…
- 请各位会员注意了
- 2007年5月，本书店…
- 我国图书市场将有很大转…
- 高远书店又到一批新书！

创意生活 漫有引力
那些与青春有关的日子
——2007年当当网夏季漫画总动员

书 店 公 告

欢迎光临高远网上书店！！
祝您购书愉快！！

贞觀長歌

2008年考研
政治理论复习导本

全国计算机等级考试
一级MS Office教程 (2004年版)

新到图书

全国计算机等级考试一…
作 者：教育部考试中心
出版社：南开大学出版社
图书定价：30.00元
高远店价：22.00元
购买 收藏

2008年考研政治理…
作 者：李淮春
出版社：中国人民大学出
图书定价：39.00元
高远店价：28.00元
购买 收藏

- 贞观长歌 77%
- 三国演义(上下)——中国古… 45%
- 周礼·仪礼 78%
- 机械设计手册（新版第1卷） 81%
- 算法导论（原书第2版） 56%
- 激烟北平 70%
- 编程卓越之道（第二卷）：运… 78%
- 西方经济学：宏观部分（第三… 79%

读者关注

全国计算机等级考试一级…
算法导论（原书第2版）
贞观长歌
2008年考研政治理论…
大明王朝1566
C++ Primer中…
深入解析Windows…
机械设计手册（新版第1…
长征
这样装修最省钱
更多排行>>

销售排行

机械设计手册（新版第1…
大明王朝1566
长征
C++ Primer中…
这样装修最省钱
管理信息系统精要（第6…
信息系统项目管理师辅导…
深入解析Windows…
盗墓笔记
西方经济学：宏观部分（…
更多排行>>

推荐图书区

- 贞观长歌 30.00元
- 三国演义(上下)——中国古典文学读本… 17.60元
- 激烟北平 21.00元
- 编程卓越之道（第二卷）：运用底层语言… 54.00元
- 机械设计手册（新版第1卷） 88.00元
- 深入解析Windows操作系统：第4… 76.00元
- 信息系统项目管理师辅导教程（上下册） 102.00元
- C++ Primer中文版(第4版)… 76.00元
- 长征 25.00元

特价淘宝

- 西部指南：12省部长纵论… 50%
- 盗墓笔记 49%



算法导论（原书第2版）
内容提要
本书深入浅出，全面地介绍了计算机算法。对每一个算法的分析既易于理解又十分有趣，并保持了数学严谨性…
图书定价：85.00 高远店价：48.00 购买



三国演义(上下)——中国…
图书定价：39.50 高远店价：17.60 购买

实惠多多 惊喜多多 点击进入

购物步骤 选书->加入购物车->去下订单->确认收货人信息->选付款方式->选配送方式->在线支付或下单后汇款->汇款确认->发货->完成

- 购书流程
- 常见问题
- 付款方式
- 交易条款
- 配送说明
- 安全和隐私
- 销售和售后服务
- 法律声明

合作网站链接： | 百 度 | google | 当当网 | 中华网

客户服务中心信箱：support@onlinebook.com 热线直拨：0298826****
公司地址：西安市高新区科技路 邮编：710066 管理登陆
Copyright © 2005-2006.All Rights Reserved .

图 10-2 网上书店网站首页



图 10-3　读者查看图书详细信息页面



订单号	合计金额	收货人	订货人	付款方式	收货方式	订单状态
2007529174717	26.00元	张三	张三	货到付款	送货上门	未作任何处理
2007529174648	46.00元	张三	张三	货到付款	送货上门	未作任何处理

图 10-4　用户登录后查看订单页面



图 10-5 系统后台图书管理页面

10.1.2 用户注册、登录和注销模块介绍

电子商务网站首先应具有的功能就是能够对访问的浏览者进行身份识别和确认，这样才能记录和跟踪用户的行为。因此要在网站与浏览者交互的页面上，提供用户注册、登录接口。访问网站的浏览者可能被网站吸引或者有购买图书的意向，这时就需要先注册为网站的合法用户，以便能够使用网站所提供的服务。注册为用户时，需要阅读网站的注册条约，只有在同意后，才可以继续。紧接着要注册人提供用户名、密码、个人信息等资料。注册成功后，用户就可以使用登录功能来使用用户权利了。例如登录后才可以购买图书或加入购物车，查看订单。

1. 填写注册信息

注册用户功能，用户需要单击图 10-6 所示的用户注册登录界面中的“注册”链接，会转到注册协议页面，如图 10-7 所示。



图 10-6 用户注册登录界面

高远网上书店注册条约

高远网上书店服务条款协议

欢迎阅读高远网上书店服务条款协议(下称“本协议”)。本协议阐述之条款和条件适用于您使用本网站所提供的各种服务(下称“服务”)。 1. 接受条款。

您通过本网站注册成为会员，即表示您同意自己已经与“高远网上书店”订立本协议，且您将受本协议的条款和条件约束

2.用户注册资格

“服务”仅供能够根据相关法律订立具有法律约束力的合约的个人或公司使用。因此，您的年龄必须在十八周岁或以上（含十八周岁），才可使用本网站服务。如不符合本项条件，请勿使用“服务”。“高远网上书店”可随时自行全权决定拒绝向任何人士提供“服务”。“服务”不会提供给被暂时或永久中止资格的“高远网上书店”会员。

3. 收费。

“高远网上书店”对本协议所规定的注册用户不收取任何费用。本网站保留在无须发出通知的情况下，暂时或永久地更改或停止部分或全部“服务”的权利。 4. “高远网上书店”网站仅作为交易地点。

5. 您授予本网站的许可使用权。 您授予本网站独家的、全球通用的、永久的、免费的许可使用权利（并有权在多个管面对该权利进行再授权），使本网站有权(全部或部分地)使用、复制、修订、改写、发布、翻译、分发、执行和展示"您的资料"或制作其派生作品。

我接受　不接受

图 10-7　用户注册协议

当用户接受注册协议后将会跳转到如图 10-8 所示的填写用户注册信息的页面。

请填写注册用户信息

用 户 名：

用于登录时使用，用户名长度不能小于4，不大于12，且使用英文字母开头。

真实姓名：

请填写真实姓名，以便发货确认。

E-Mail ：

请填写您有效的邮件地址，以便于我们为您提供安全有效的服务。

密　码：

长度必须大于6个字符小于16个字符，只能为英文、数字，例如：temp3000等。

确认密码：

请将输入的密码再次输入确认，以便于您记忆。

密码提问：(用于密码遗忘时使用)

密码答案：(密码遗忘时将验证此答案重设密码)

提　交

图 10-8　填写用户注册信息

2. 提交注册信息过程

如果用户的注册信息内容合法，那么将会显示如图 10-9 所示的用户注册成功页面。

用户注册成功

恭喜huangrong，您已注册成为《高远网上书店》正式用户，请进行下一步操作：

- 填写收货人详细资料
- 返回书店首页

图 10-9　显示注册成功

如果注册输入的用户名或 E-mail 邮箱已经注册过，则会显示如图 10-10 所示信息。

3. 用户登录

用户登录界面对话框如图 10-6 所示。登录成功会变为图 10-11 所示。

用户注册失败

- 您输入的用户名或E-mail地址已存在，请返回重新输入!
- 点击返回上一页

图 10-10　注册失败提示

4. 用户注销

对于 Web 环境下的管理系统,用户如果不再操作了,一定要安全地退出系统,不能简单地关闭浏览器,防止用户的会话在缓存中存在,被人利用产生不必要的麻烦。单击登录界面中的“注销登录”链接,直至显示如图 10-12 所示提示信息。

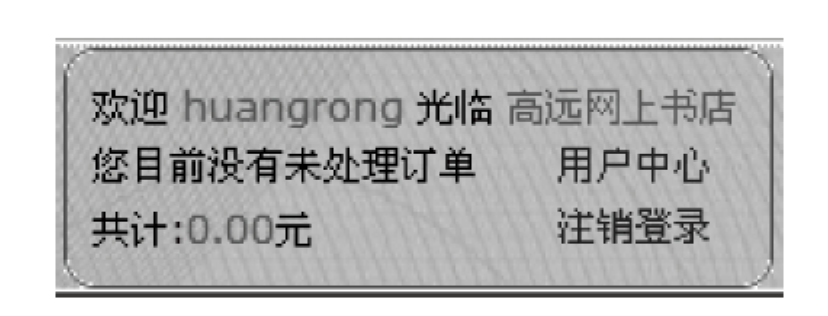


图 10-11　登录成功信息



图 10-12　注销登录成功

10.2　测试计划

测试计划是整个测试过程的重要组成部分,一般由测试项目经理来完成。测试计划只有预算、人员安排和时间进度还远远不够,要涉及许多测试工作的具体规划。很难想象,一个没有经过很好策划的测试项目能够顺利进展。

测试计划工作的提交成果是一份完整的测试计划报告。测试计划报告的模板不必千篇一律,要随着软件的应用行业、软件功能以及性能要求、管理规范性要求等而定。但一个完整的测试计划一般均包括被测试项目的背景、测试目标、测试的范围、方式、资源、进度安排、测试人员组织以及与测试有关的风险等方面。

10.2.1　概述

本测试项目拟对高远网上书店管理系统进行测试。

测试的目标是要找出影响高远网上书店管理系统正常运行的错误,分别在功能、性能、安全等方面检验系统是否达到相关要求。

本次集成测试同时采用黑盒和白盒测试技术(重点在黑盒测试上)。测试手段为手工与自动测试相结合。

本测试计划面向相关项目管理人员、测试人员和开发人员。

10.2.2　定义

- 质量风险:被测试系统不能实现描述的产品需求或者系统不能达到用户的期望的行为,即系统可能存在的错误。

- 测试用例：为了查找被测试软件中的错误而设计的一系列的操作数据和执行步骤，即一系列测试条件的组合。
- 测试工具：应用于测试用例的硬件/软件系统，用于安装或撤销测试环境、创造测试条件、执行测试或者度量测试结果等工作。测试工具独立于测试用例本身。
- 进入标准：一套标准，用于决定项目是否可以退出当前的测试阶段、进入下一个测试阶段或者结束项目。与进入标准类似，测试过程后几个阶段的退出标准一般很苛刻。
- 功能测试：集中于功能正确性方面的测试。功能测试必须和其他测试方法一起处理潜在的重要的质量风险，比如性能、负荷、容积和容量等。

10.2.3　质量风险摘要

质量风险摘要如表 10-1 所示。

表 10-1　质量风险摘要表

风险编号	潜在的故障模式	故障的潜在效果	危险性	影响	优先级
1	各模块功能无法顺利实现	无法登录、注册不成功等	4	5	5
2	数据处理	购物车费用计算不准确	5	4	5
		注册信息不准确	3	3	4
3	并发控制	多用户访问时，系统出现速度低等问题	5	3	4
4	错误处理	不能阻止错误发生；错误发生后处理不当	4	3	4
5	界面不友好	没有必要的提示；操作不方便	1	5	2
6	系统响应速度慢	对用户提交信息的响应和处理速度慢	1	5	3
……	……	……	……	……	……

- 危险性：表示故障对系统影响的大小。5——致命的，4——严重的，3——一般，2——轻微，1——无。
- 影响：5——一定影响所有用户，4——可能影响一些用户，3——对有些用户可能影响，2——对少数用户有限影响，1——在实际使用中难以察觉的影响。
- 优先级：表示风险可以被接受的程度。5——很紧急，必须马上纠正，4——不影响进一步测试，但必须修复，3——系统发布前必须修复，2——如果时间允许应该修复，1——最好修复。

10.2.4　测试进度计划

测试进度计划表如表 10-2 所示。

表 10-2 测试进度计划表

阶　段	任务号	任务名称	前序任务号	工时(人日)	提交结果
测试计划	1	制定测试计划		3	测试计划
测试系统开发与配置	2	人员安排	1	0.5	任务分配
	3	测试环境配置 开发问题记录工具 建立问题数据库	1,2	3	可运行系统的环境 问题记录工具 问题记录数据库
	4	测试用例设计 测试数据恢复工具设计开发	1,2	30	测试用例 数据恢复工具
测试执行	5	第 1 阶段测试通过	1,2,3,4	30	测试结果记录
	6	第 2 阶段测试通过	5	20	测试结果记录
	7	第 3 阶段测试通过	6	10	测试结果记录
测试总结分析	8	推出系统测试	7	4	测试分析报告

10.2.5 进入标准

(1)“测试小组”配置好软硬件环境,并且可以正确访问这些环境。

(2)“开发小组”已完成所有特性和错误修复并完成修复后的单元测试。

(3)“测试小组”完成“冒烟测试”——程序包能打开,随机的测试操作正确完成。

10.2.6 退出标准

(1)“开发小组”完成了所有必须修复的错误。

(2)“测试小组”完成了所有计划的测试。没有优先级为 3 以上的错误。优先级为 2 以下的错误少于 5 个。

(3)“项目管理小组”认为产品实现稳定且可靠。

10.2.7 测试配置和环境

- 服务器一台。
- 客户机 5 台。
- 打印机一台。
- 地点:软件工程实验室。

10.2.8 关键参与者

- 测试经理。
- 测试人员。
- 开发人员。

• 项目管理人员。

10.3　测试过程概述

广义地说，测试工作贯穿了一个软件项目开发过程的始终，从项目的策划和相关文档生成直到软件通过用户的验收。通常所说的测试是指运行软件系统（或者单个模块）以验收其是否满足用户要求的过程。

对该网上书店管理系统的测试按照一般测试过程，将其分为单元测试、集成测试、系统测试和验收测试 4 个阶段。

10.3.1　单元测试

单元测试常常是动态测试和静态测试两种方式并举的。动态测试可由开发人员去运行局部功能或模块以发现系统潜藏的错误，也可以借助测试工具去测试。静态测试即是代码审查。审查的内容包括代码规则和风格、程序设计和结构、业务逻辑等。

网上书店管理系统中涉及到一些费用计算问题，逻辑性很强，需要程序结构也很复杂。面对复杂的业务流程，面对管理各异的用户需求，没有白盒测试是不可想象的。开发人员就要严格地依照系统设计去检查代码的逻辑结构，选取有代表性的测试用例去测试相关的模块。

10.3.2　集成测试

集成测试（有时被分为集成测试和确认测试两个阶段）是指将各模块组装起来进行测试，以检查与设计相关的软件体系结构的有关问题，并确认软件是否满足需求规格说明书中确定的各种需求。

这个阶段的测试需要一个完备的测试管理过程。集成测试过程可以分为测试准备、测试计划、测试设计、测试执行和测试总结 5 个阶段。

测试准备阶段是指测试人员准备测试资源，熟悉系统的过程。

测试计划阶段包含制定测试策略、资源分配、风险预警和进度安排等内容，此项工作由测试负责人来做。

测试设计阶段包括设计测试用例及相关管理工具的设计。

完成测试设计工作后，就开始执行实际的测试工作了。

测试时另外一项非常重要的工作就是做好系统缺陷记录。

对经过修改后的系统再次测试即是回归测试。

测试结束后要及时总结分析测试结果。测试结果的总结与分析一方面是为了提供一个系统功能、性能和稳定性等方面的完整的分析和结论，另外要对测试过程本身做出总结，总结成功的经验和失败的教训，以使日后的工作开展得更顺利。

10.3.3 系统测试

系统测试是在真实或模拟系统运行的环境下，检查完整的程序系统能否和系统（包括硬件、外设、网络和系统软件、支持平台等）正确配置、连接，并满足用户需求。

系统测试也应该经过测试准备、测试计划、测试设计、测试执行和测试总结5个阶段，每个阶段所做工作内容与集成测试很相似，只是关注点有所不同。

在网上书店管理系统的系统测试中，要搭建更真实的运行环境，另外还要在不同的操作系统下进行测试，如数据库服务器分别搭建在UNIX环境和Windows NT环境下长时间多客户端并发运行系统的各项功能，并观测服务器的承受能力（系统的反应时间、服务器的资源占用情况等）。

10.3.4 验收测试

验收测试是指在用户对软件系统验收之前组织的系统测试。测试人员都是真正的用户，在尽可能真实的环境下进行操作，并将测试结果进行汇总，由相关管理人员对软件做出评价以及是否验收的决定。

网上书店管理系统一般在用户验收之前都需要对系统进行一段时间的试运行，因此可以说该系统的验收测试就是实际的使用（但用户一般要参与软件的系统测试，即所谓的β测试，不然用户是不会放心让系统试运行的）。

因为验收测试由用户完成，不同软件实际应用的差异性又很大，这里就不对其详加论述了。

10.4 测试用例设计

测试用例应该由测试人员在充分了解系统的基础上在测试之前设计好，测试用例的设计是测试系统开发中一项非常重要的内容。在集成测试阶段，测试用例的设计依据为系统需求分析、系统用户手册和系统设计报告等相关资料，测试人员要和开发人员充分交互。另外还有一些内容由测试人员的相关背景知识、经验和直觉等产生。

测试用例的设计需要考虑周全。在测试系统功能的同时，还要检查系统对输入数据（合法值、非法值和边界值）的反应，要检查合法的操作和非法的操作，检查系统对条件组合的反应等。好的测试用例可让其他人能够很好地执行测试，能够快速地遍历所测试的功能，能够发现至今没有发现的错误。所以测试用例应该由经验丰富的系统测试人员来编写，而新手则应该多阅读一些好的测试用例，并且在测试实践中用心体会。

对于高远网上书店管理系统，我们针对该系统的用户注册、登录和注销模块进行测试用例的设计。测试内容如下：

- 注册。
- 登录。
- 注销。

1. 注册模块的测试用例举例(见表 10-3)

表 10-3 测试用例——注册

测试配置		测试数据					测试结果
测试用例编号	测试用例名称	用户名	密码	确认密码	电子邮件	期望结果	
01	用户名为英文	marry	123456	123456	marry@163.com	提交成功	
02	用户名为中文	王涛	123456	123456	wt@qq.com	提交成功	
03	用户名小于 4 个字符	lee	123456	123456	lee@163.com	提示“用户名不能小于 4 个字符”	
04	用户名为空		123456	123456	lee@163.com	提示“用户名不能为空”	
05	用户名超过 12 个字符	Abcedfg Higklmn	123456	123456	lee@163.com	提示“用户名不能超过 12 个字符”	
……	……	……	……	……	……	……	

2. 登录模块测试用例举例(见表 10-4)

表 10-4 测试用例——登录

测试配置		测试数据				测试结果
测试用例编号	测试用例名称	用户名	密码	验证码	期望结果	
01	密码错误	marry	456789	2013	提示“密码错误”	
02	用户名错误	maarry	123456	2013	提示“用户名错误”	
03	验证码错误	lee	123456	2008	提示“验证码错误”	
04	均正确	marry	123456	2013	登录成功	
05	用户名为空		123456	2013	提示“用户名不能为空”	
……	……	……	……	……	……	

3. 注销模块测试用例举例(见表 10-5)

表 10-5 测试用例——注销

测试配置		测试数据				测试结果
测试用例编号	测试用例名称	操作描述			期望结果	
01	左键单击	左键单击“注销按钮”			弹出“注销成功”对话框	
02	左键双击	左键双击“注销按钮”			提示“操作有误”	
03	右键单击	右键单击“注销按钮”			提示“操作有误”	
……	……	……	……	……	……	

以上是以用户注册、登录和注销模块为例进行测试用例设计的举例，读者可以完善其他设计用例，这里不再详细描述。

10.5 缺陷报告

在测试执行阶段，需要利用缺陷报告来记录、描述和跟踪被测试系统中已经被捕获的不能满足用户对质量的合理期望的问题，即缺陷或者错误。缺陷报告可以采用多种样式，可以利用 Word、Excel 等，这要根据系统的复杂程度而定。如果需要灵活地、交互地存储、操作、查询、分析和报告大量数据，还是需要数据库的。错误跟踪数据库可以自己开发也可以购买。

测试人员、系统开发人员和相关问题评审人员如何打开、读取和写入缺陷报告数据库的形式并不重要，重要的是对于问题的描述应该是完整的、严谨的、简洁的、清晰的和准确的。

这里列出编写好的错误报告的几个要点。

(1) 再现：尽量三次再现故障。如果问题间断的，则要报告问题的发生频率。

(2) 隔离：确定可能影响错误再现的变量，例如配置变化、工作流、数据集，这些都可能改变错误的特征。

(3) 推广：确定系统其他部分是否可能出现这种错误，特别是那些可能存在更加严重特征的部分。

(4) 压缩：精简任何不必要的信息，特别是冗余的测试步骤。

(5) 去除歧义：使用清晰的语言，尤其要避免使用那些有多个不同或相反含义的词汇。

(6) 中立：公正表达自己的意思，对错误及其特征的事实进行陈述，避免夸张、幽默和讽刺。

(7) 评审：至少有一个同行，最好是一个有丰富经验的测试工程师或测试经理，在你递交错误报告之前读一遍。

10.6 测试结果总结分析

一个阶段的系统测试结束以后，应该对系统有一个完整的测试总结报告，给出系统最终测试后在功能、性能等方面所达到的状态的总结和评价，通常测试总结报告要包含量化的描述。测试总结报告将呈现给测试部门、开发部门以及公司的相关负责人。

10.6.1 测试总结报告

图 10-13 所示的是测试总结报告的一个模板，各行业、各阶段的软件测试会有具体不同的总结报告，但基本上应该有本模板所展示的项目。

*** 测试报告	
项目编号：	项目名称：
项目软件经理：	测试负责人：
测试时间：	
测试目的与范围：	
测 试 环 境	
名　　称	软 件 版 本
服务器操作系统	
数据库	
应用服务器	
测试软件	
测试机操作系统	
测试数据说明：	
总体分析：	
典型性具体测试结果：	

图 10-13　测试总结报告模板

10.6.2　测试用例分析

对工作的及时总结，有利于及时调整方向，大大提高工作效率。测试工作的效果直接依赖于测试用例的编写和执行状况，所以在测试过程中和测试结束后都要对关于测试用例的一些重要值进行度量。

关于测试用例的分析，通常包括以下的内容：

- 计划了多少个测试用例？实际运行了多少？
- 有多少测试用例失败了？
- 在这些失败的测试用例中，有多少个在错误得到修改后最终运行成功了？
- 这些测试平均占用的运行时间比预期的长还是短？
- 有没有跳过一些测试？如果有，为什么？
- 测试覆盖了所有影响系统性能的重要事件吗？

这些问题都可以从相关的测试用例的设计和测试问题记录中找到相应的答案。当然，如果使用了数据库，这些问题就更能轻松地被解答了。测试用例的分析报告可以以多种形式体现出来：文字描述、表、图等。

10.7 软件测试自动化工具

实际测试需要投入大量的时间和精力，测试工作同样也可以采用其他领域和行业中运用多年的办法——开发和使用工具，即自动化测试使工作更加轻松和高效。采用测试工具不但能提高效率，节约成本，还可以模拟许多手工无法模拟的真实场景。

在本章测试项目中，我们使用 LoadRunner 来进行性能测试。

启动 Visual User Generator，通过菜单 File→New 新建一个用户脚本，如图 10-14 所示。在弹出的菜单中选择合适的通信协议，如图 10-15 所示。

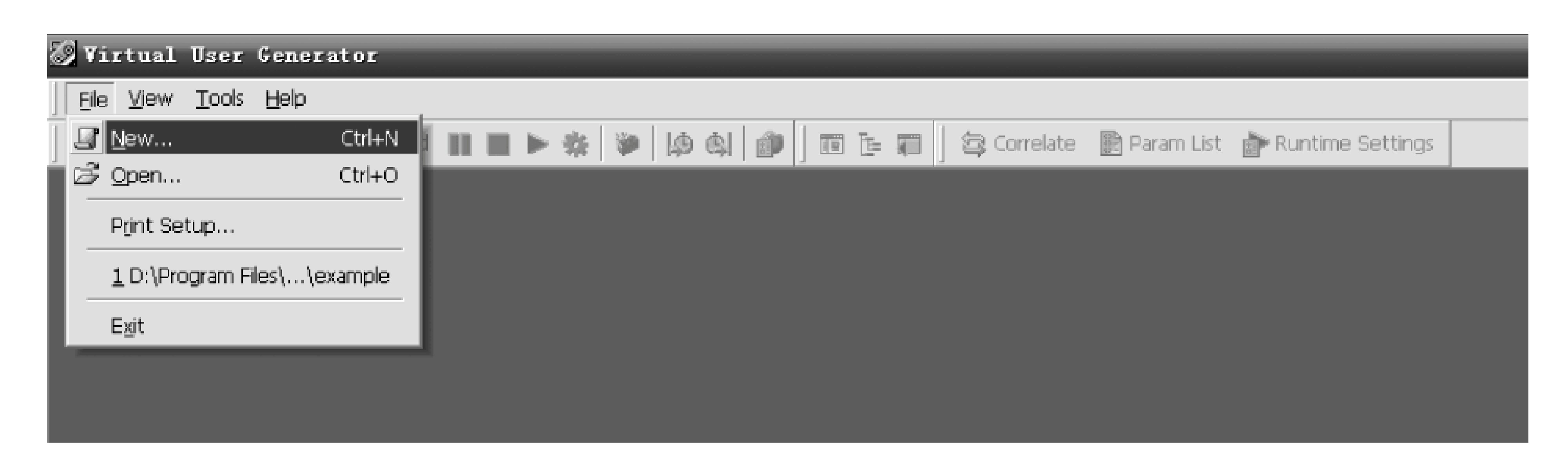


图 10-14 LoadRunner 新建页面

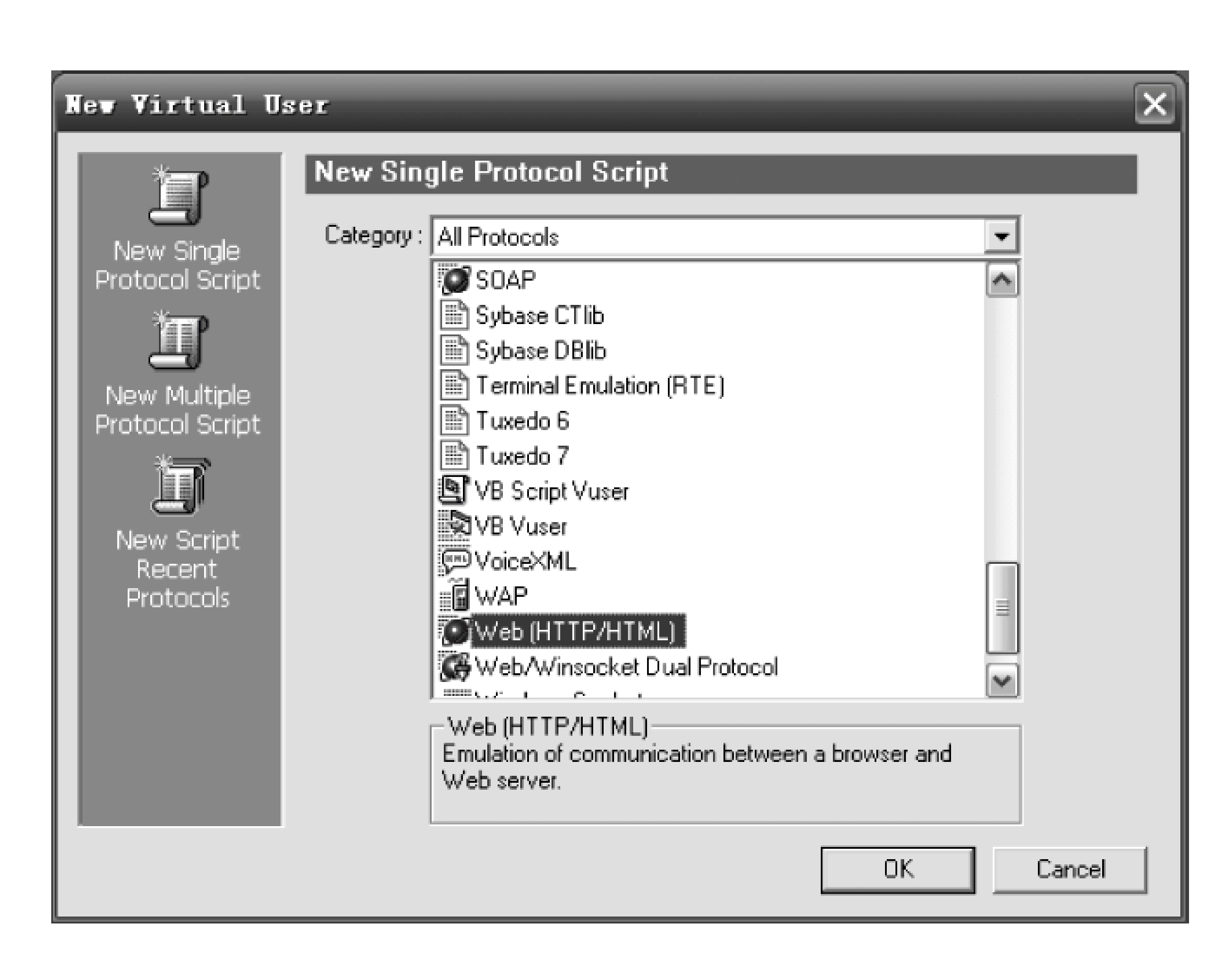


图 10-15 LoadRunner 协议选择页面

在菜单栏中选择 Vuser→Start Recording 或者在工具栏中单击相应按钮，都可以启动录制脚本的命令，打开录制窗口。图 10-16 所示是录制后自动生成的某一段脚本。

接下来运行测试脚本，执行“运行”命令，VuGen 先编译脚本，检查是否有语法等错误。如果有错误，VuGen 将会提示错误。双击错误提示，VuGen 能够定位到出现错误的那一行。为了验证脚本的正确性，还可调试脚本，如在脚本中加断点等，如图 10-17 所示。图 10-18 是测试后生成的 LoadRunner 结果分析图。

Virtual User Generator - [flight - Web (HTTP/HTML)]

File Edit View Insert Vuser Actions Tools Window Help

Start Record　Correlate　Param List　Runtime Settings

flight - Web (HTTP/HTML)

vuser_init
Action
Action1
Action2
Action3
Action4
vuser_end

```
Action1()
{
    web_url("welcome.pl",
        "URL=http://localhost/MercuryWebTours/welcome.pl?page=search",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://localhost/MercuryWebTours/nav.pl?page=menu&in=home",
        "Snapshot=t5.inf",
        "Mode=HTML",
        EXTRARES,
        "Url=images/sun_with_logo.gif", "Referer=http://localhost/MercuryWebTours/nav.pl?page=menu&in=fligh
        LAST);

    lr_think_time( 21 );

    web_submit_form("reservations.pl",
        "Snapshot=t6.inf",
        ITEMDATA,
        "Name=depart", "Value=London", ENDITEM,
        "Name=departDate", "Value=07/28/2006", ENDITEM,
        "Name=arrive", "Value=Los Angeles", ENDITEM,
        "Name=returnDate", "Value=07/29/2006", ENDITEM,
        "Name=numPassengers", "Value=1", ENDITEM,
        "Name=roundtrip", "Value=on", ENDITEM,
        "Name=seatPref", "Value=Window", ENDITEM,
        "Name=seatType", "Value=First", ENDITEM,
        "Name=findFlights.x", "Value=77", ENDITEM,
        "Name=findFlights.y", "Value=18", ENDITEM,
        LAST);

    lr_think_time( 17 );

    web_submit_form("reservations.pl_2",
        "Snapshot=t7.inf",
        ITEMDATA,
        "Name=outboundFlight", "Value=231;1291;07/28/2006", ENDITEM,
        "Name=returnFlight", "Value=320;1449;07/29/2006", ENDITEM,
```

For Help, press F1.　Col:1　Line:1　INS

图 10-16　LoadRunner 自动生成测试脚本代码

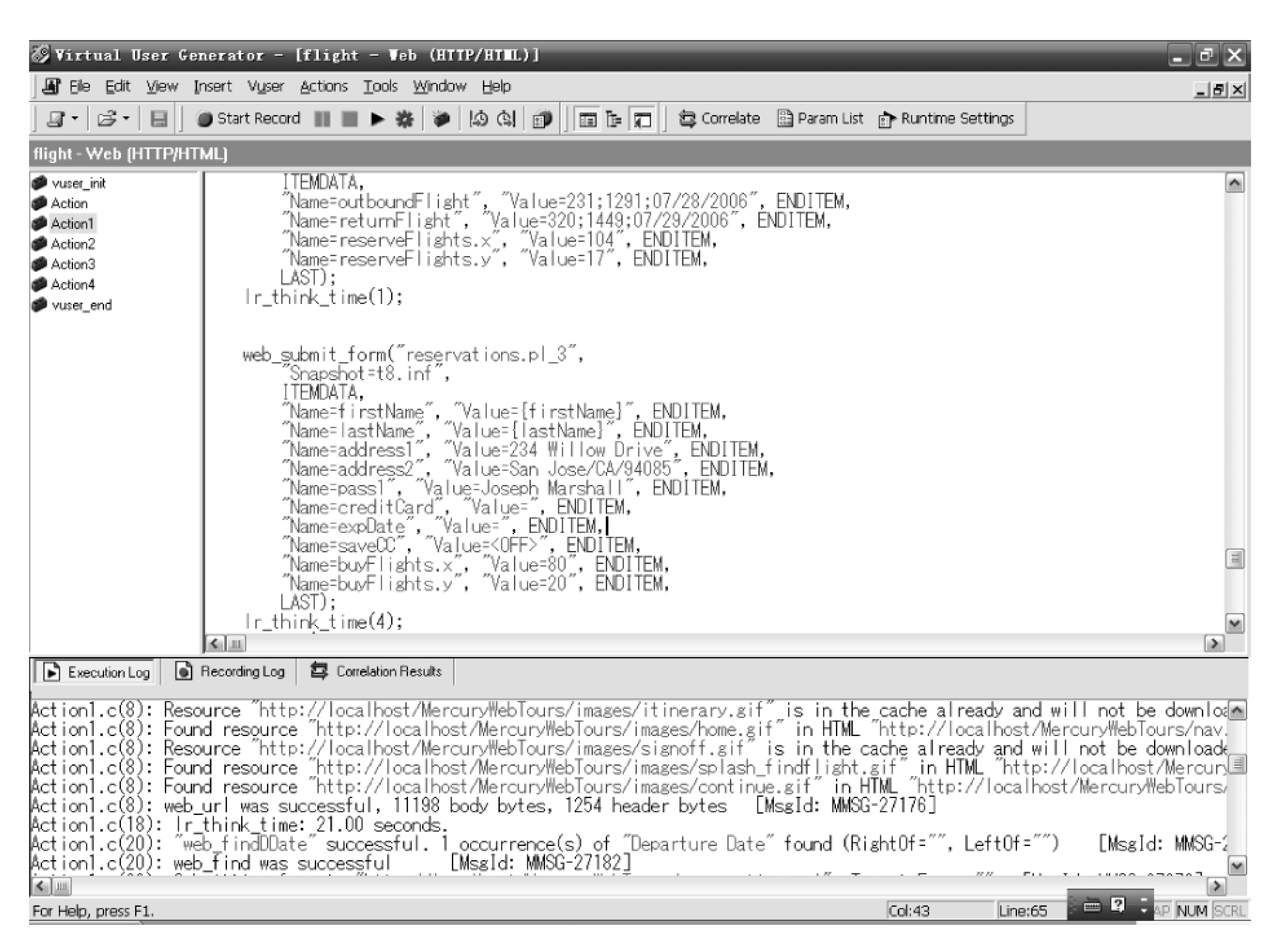


图 10-17　LoadRunner 运行测试脚本

最终可以通过 LoadRunner 的结果分析工具进行分析，如图 10-18 所示。

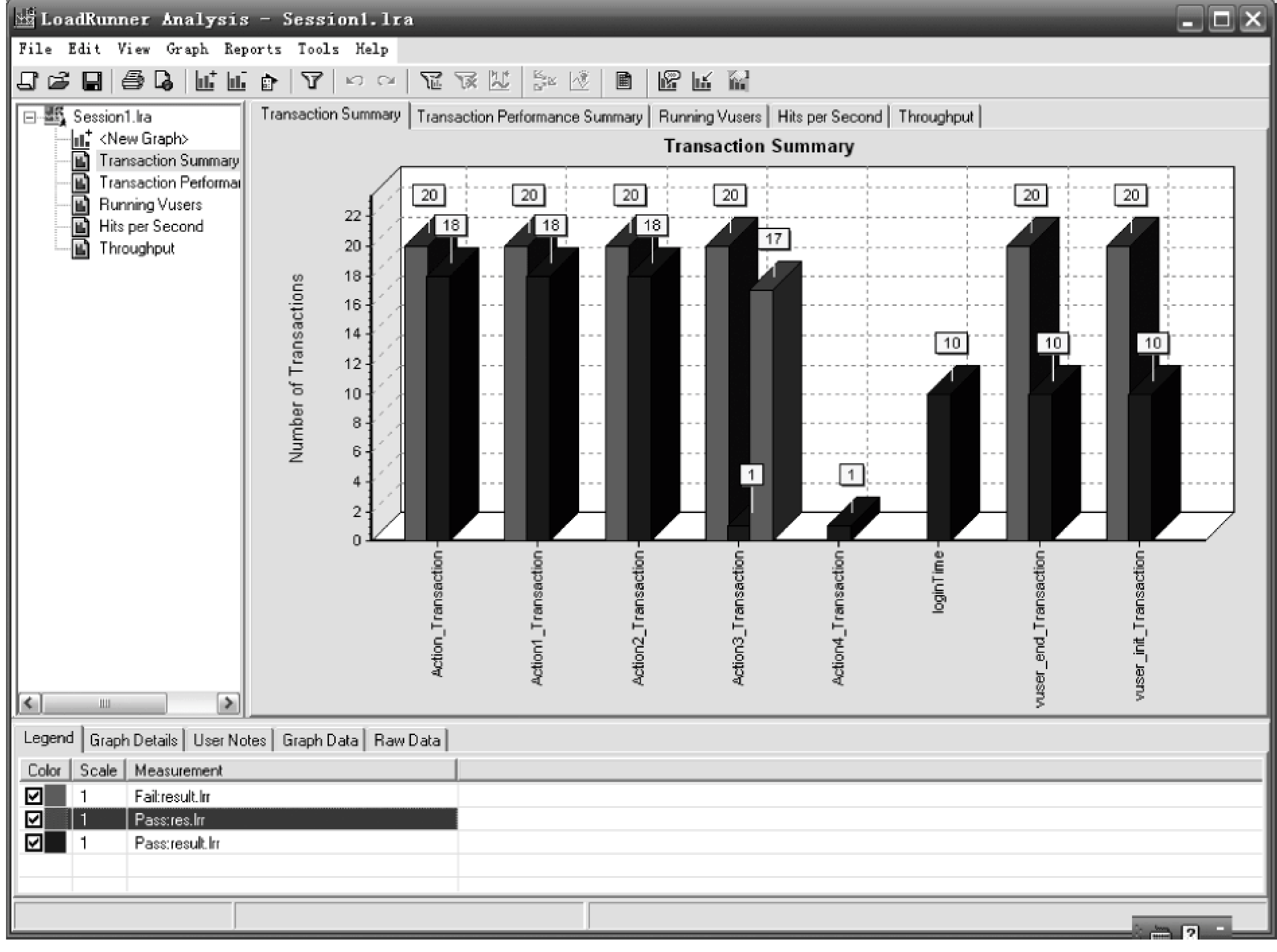


图 10-18　LoadRunner 结果分析图

10.8　文档测试

广义地说，文档测试也是软件测试的一项内容。文档测试包括对系统需求分析说明书、系统设计报告、用户手册以及与系统相关的一切文档、管理文件的审阅、评测。

系统需求分析和系统设计说明书中的错误将直接带来程序的错误；而用户手册将随着软件产品交付用户使用，是产品的一部分，也将直接影响用户对系统的使用效果，所以任何文档的表述都应该清楚、准确，含糊不得。

文档测试时应该慢慢仔细阅读文字。特别是用户手册，应完全根据提示操作，将执行结果与文档描述进行比较。不要任何假设，耐心补充遗漏的内容，耐心更正错误的内容和表述不清楚的内容。

10.9　本章小结

本章以高远网上书店管理系统这个项目为实际案例，介绍了该系统的背景和功能模块，描述了测试计划的设计与书写，对测试过程进行了概述，并以该系统中的登录、注册和注销模块为例设计了测试用例，编写缺陷报告，对测试结果进行总结分析，使用 LoadRunner 进行自动化性能测试并对其结果进行分析，将一个较为完整的测试流程展现在了读者面前。

习题 10

参考本章的相关步骤和文档，找一个你所熟悉的软件系统为其定制测试计划，设计测试用例，按照测试用例执行测试并记录测试过程和做好测试结果的分析。

参 考 文 献

[1] 陈汶滨,朱小梅,任冬梅. 软件测试技术基础[M]. 北京:清华大学出版社,2008.
[2] 郑人杰,许静,于波. 软件测试[M]. 北京:人民邮电出版社,2011.
[3] 徐光侠,韦庆杰.软件测试技术教程[M].北京:人民邮电出版社,2011.
[4] [美] Ron Patton.软件测试[M].张小松,王钰,曹跃,译.北京:机械工业出版社,2006.
[5] 周元哲.软件测试基础[M].西安:西安电子科技大学出版社,2010.
[6] 朱少民.软件测试[M].北京:人民邮电出版社,2009.
[7] 赵瑞莲.软件测试[M].北京:高等教育出版社,2004.
[8] 李幸超.实用软件测试[M].北京:电子工业出版社,2006.
[9] 贺平.软件测试教程[M].北京:电子工业出版社,2005.
[10] 朱少民.软件测试方法和技术[M].北京:清华大学出版社,2005.
[11] 马瑟.软件测试基础教材[M].北京:机械工业出版社,2008.
[12] 51Testing 软件测试网[DB/OL]. http://www.51testing.com/html/index.html.